User's Manual

RLIN3 モジュール
Software Integration System

ユーザーズ・マニュアル　Rev 1.00

本資料に記載の全ての情報は本資料発行時点のものであり、ルネサス エレクトロニクスは、
予告なしに、本資料に記載した製品または仕様を変更することがあります。
ルネサス エレクトロニクスのホームページなどにより公開される最新情報をご確認ください。

ルネサス エレクトロニクス
www.renesas.com

Rev 1.00　R01AN6335JJ0100

このマニュアルの使い方

対 象 者　このマニュアルは、LIN2.1 ソフトウェア・ドライバの機能を理解し、ドライバを使用したアプリケーションを開発、設計するユーザを対象としています。

目　　的　このマニュアルは、LIN 2.1 ソフトウェア・ドライバの機能、使い方をユーザに理解していただくことを目的としています。
なお、LIN2.1 の仕様に関しては、や ISO から発行されている LIN Specification Package Revision 2.1 を参照してください。

構　　成　このマニュアルでは，大きく分けて次の内容で構成しています。

- 製品概要
- インストレーション
- システム構築
- コンフィギュレーション
- LIN2.1 ソフトウェア・ドライバ関数

読 み 方　このマニュアルを読むにあたっては，電気，論理回路，マイクロコントローラの一般知識を必要とします。
LIN2.1 ソフトウェア・ドライバの機能を理解しようとするときは、目次に従ってお読みください。

凡　　例

データ表記の重み	：左が上位桁，右が下位桁
アクティブ・ロウの表記	：$\overline{\times\times\times}$（端子，信号名称に上線）
注	：本文中につけた注の説明
注意	：気をつけて読んでいただきたい内容
備考	：本文の補足説明
数の表記	：2 進数…××××または××××B
	10 進数…××××
	16 進数…××××H

2 のべき乗を示す接頭語（アドレス空間、メモリ容量）：
K（キロ）…$2^{10} = 1024$
M（メガ）…$2^{20} = 1024^2$
G（ギガ）…$2^{30} = 1024^3$

データ・タイプ：
ワード…32 ビット
ハーフ・ワード…16 ビット
バイト…8 ビット

すべての商標および登録商標は、それぞれの所有者に帰属します。
EEPROMは，ルネサスエレクトロニクス株式会社の登録商標です。
Windows 10は，米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。
PC/ATは，米国IBM社の商標です。
SuperFlashは，米国Silicon Storage Technology, Inc.の米国，日本などの国における登録商標です。

注意：本製品は Silicon Storage Technology, Inc.からライセンスを受けた SuperFlash®を使用しています。

目　次

第 1 章 製品概要 10
1. 1 概要 10
1. 2 特徴 10
1. 2. 1 高い移植性 10
1. 2. 2 コンフィギュレーション・ツールの提供 10
1. 3 LIN2.1 ソフトウェア・ドライバの種類 11
1. 4 実行環境 12
1. 4. 1 対象マイコン 12
1. 4. 2 メモリ容量 13
1. 4. 3 利用資源 14
1. 5 開発環境 16
1. 6 制約事項 17
1. 6. 1 クロック、ボー・レート設定 17
1. 6. 2 割り込みに関する制約事項 17
1. 6. 3 その他の制約事項 18
第 2 章 インストレーション 19
2. 1 概要 19
2. 2 インストール 19
2. 2. 1 LIN コンフィギュレータのインストール 19
第 3 章 システム構築概要 20
3. 1 LIN2.1 ソフトウェア・ドライバの位置づけ 20
3. 2 システム構築手順 21
3. 2. 1 LIN コンフィギュレータによるファイル生成 22
3. 2. 2 ユーザ・アプリケーション 23
3. 2. 3 ビルド 24
第 4 章 スマート・コンフィグレータとの連携 25
4. 1 操作手順 26
4. 2 コンフィグ値 27
4. 2. 1 LIN コンフィギュレータ用設定 27
4. 2. 2 LIN ドライバ用設定 28
第 5 章 LIN アプリケーションビルド方法 30
5. 1 LIN ドライバ・ライブラリ生成 31
5. 1. 1 ライブラリ用コンパイラ・オプション 32
5. 1. 2 confmlin_opt.h 編集（マスター用） 33
5. 1. 3 confslin_opt.h 編集（スレーブ用） 34
5. 1. 4 IAR I/O ヘッダ・ファイル指定 35
5. 2 LIN アプリケーション・プログラム作成 36
5. 2. 1 開発環境プロジェクト・ファイル作成 37
5. 2. 2 チャネル専用ソース・コード作成 37
5. 2. 3 周辺ハードウェア処理実装 38
5. 2. 4 コンパイラ・オプション編集（conflin_x.h） 40

5. 2. 5 公開定数編集（conflin_x.c） 41
5. 2. 6 スケジューラの実装（マスターのみ） 42
5. 2. 7 ユーザ定義コール・アウトの実装 44
5. 2. 8 使用セクションの設定 45
第 6 章 LIN コンフィギュレータ 47
6. 1 概要 47
6. 1. 1 特徴 47
6. 1. 2 実行環境 47
6. 1. 3 出力フォルダ 47
6. 2 ファイル生成手順 48
6. 2. 1 LIN コンフィギュレータの起動 49
6. 2. 2 新規コンフィギュレーションの開始 52
6. 2. 3 デバイス選択 53
6. 2. 4 チャネル設定 55
6. 2. 5 ボー・レート設定 57
6. 2. 6 メッセージ管理 58
6. 2. 7 スケジュール管理 64
6. 2. 8 ノード設定 66
6. 2. 9 その他設定 67
6. 2. 10 設定ファイルの保存、読出し 67
6. 2. 11 ソース・コード出力 68
6. 3 エラー・メッセージ一覧 69
6. 4 ワーニング・メッセージ一覧 72
第 7 章 LIN2.1 ソフトウェア・ドライバ概要 73
7. 1 シグナルタイプ 【マスター/スレーブ】 73
7. 2 フレーム構成 【マスター/スレーブ】 74
7. 2. 1 バイト・フィールド 75
7. 2. 2 Break フィールド 75
7. 2. 3 フレーム長 75
7. 3 フレーム転送 【マスター/スレーブ】 77
7. 3. 1 無条件フレーム転送 77
7. 3. 2 イベント・トリガ・フレーム転送 78
7. 3. 3 スポラディック・フレーム転送 81
7. 4 レスポンス・エラー通知機能 【マスター/スレーブ】 82
7. 5 スリープ・ウェイクアップ機能 【マスター/スレーブ】 83
7. 5. 1 スリープ機能 83
7. 5. 2 ウェイクアップ機能 83
7. 6 ノード・コンフィギュレーション機能 【マスター/スレーブ】 85
7. 6. 1 ノード情報 85
7. 6. 2 ノード・コンフィギュレーション 86
7. 7 スケジューリング機能 【マスター】 88
7. 7. 1 スケジュール遷移（l_sch_tick） 88
7. 7. 2 スケジュール切り替え（l_sch_set） 90

7. 8 ボー・レート自動検出機能【スレーブ】 91
7. 9 ドライバ・コンフィグレーション 92
7. 9. 1 スレーブ・ドライバ・コンフィグレーション 92
7. 9. 2 マスター・ドライバ・コンフィグレーション 96
第 8 章 LIN2.1 ソフトウェア・ドライバ関数（スレーブ用） 99
8. 1 LIN2.1 ソフトウェア･スレーブ・ドライバ関数一覧 99
8. 2 データ・タイプ（スレーブ用） 100
8. 3 LIN2.1 ソフトウェア・スレーブ・ドライバ関数の説明 101
8. 3. 1 [スレーブ] LIN2.1 ソフトウェア・ドライバとクラスタ管理（1 種類） 103
8. 3. 2 [スレーブ] スカラ・シグナル読み出し（3 種類） 105
8. 3. 3 [スレーブ] スカラ・シグナル書き込み（3 種類） 111
8. 3. 4 [スレーブ] バイト・アレイ読み出し（1 種類） 116
8. 3. 5 [スレーブ] バイト・アレイ書き込み（1 種類） 119
8. 3. 6 [スレーブ] 通知（2 種類） 122
8. 3. 7 [スレーブ] インタフェース・マネージメント（3 種類） 125
8. 3. 8 [スレーブ] ユーザ定義コール・アウト（4 種類） 131
第 9 章 LIN2.1 ソフトウェア・ドライバ関数（マスター用） 136
9. 1 LIN2.1 ソフトウェア・マスター・ドライバ関数一覧 136
9. 2 データ・タイプ（マスター用） 137
9. 3 LIN2.1 ソフトウェア・マスター・ドライバ関数の説明 138
9. 3. 1 [マスター] LIN2.1 ソフトウェア・ドライバとクラスタ管理（1 種類） 140
9. 3. 2 [マスター] スカラ・シグナル読み出し（3 種類） 142
9. 3. 3 [マスター] スカラ・シグナル書き込み（3 種類） 148
9. 3. 4 [マスター] バイト・アレイ読み出し（1 種類） 153
9. 3. 5 [マスター] バイト・アレイ書き込み（1 種類） 156
9. 3. 6 [マスター] 通知（2 種類） 159
9. 3. 7 [マスター] スケジュール・マネージメント（2 種類） 162
9. 3. 8 [マスター] インタフェース・マネージメント（4 種類） 167
9. 3. 9 [マスター] ノード・コンフィギュレーション（4 種類） 175
9. 3. 10 [マスター] ユーザ定義コール・アウト（4 種類） 184
第 10 章 LIN 記述ファイル（LDF）の記述例 189
付録　改版履歴 193

表　目　次

表 1-1　ノード情報一覧 11
表 1-2　対象マイコン一覧（スレーブ） 12
表 1-3　対象マイコン一覧（マスター） 12
表 1-4　メモリ容量（スレーブ） 13
表 1-5　メモリ容量（マスター） 13
表 1-6　利用資源一覧(スレーブ) 14
表 1-7　利用資源一覧(マスター) 15
表 1-8　ソフトウェア一覧 16
表 1-9　使用条件一覧 17
表 4-1　LIN コンフィギュレータ用定義値 (r_rlin3_config.h) 27
表 4-2　LIN ドライバ用定義値 (r_rlin3_config.h) 28
表 5-1　ライブラリ用コンパイラ・オプション一覧 32
表 5-2　コンパイラ・オプション（conflin_x.h）一覧 40
表 5-3　LIN2.1 ソフトウェア・ドライバ使用セクション（CC-RL） 45
表 5-4　LIN2.1 ソフトウェア・ドライバ使用セクション（IAR） 46
表 6-1　ツリー項目、ボタン、Tool メニューの操作(マスターチャネル時) 51
表 6-2　ツリー項目、ボタン、Tool メニューの操作(スレーブチャネル時) 51
表 6-3　エラー・メッセージ一覧 69
表 6-4　ワーニング・メッセージ一覧 72
表 7-1　フレーム構成要素一覧 74
表 7-2　フレーム種類一覧 77
表 7-3　レスポンス・エラー一覧 82
表 7-4　ノード情報一覧 85
表 7-5　マスター・リクエスト・フレーム・フォーマット　(Assign frame identifier range) 86
表 7-6　マスター・リクエスト・フレーム・フォーマット (Read by identifier) 87
表 7-7　スレーブ・レスポンス・フレーム・フォーマット 87
表 7-8　否定応答フォーマット 87
表 7-9　ドライバ・コンフィグレーション(RL78/F23,F24 スレーブ) (1/2) 92
表 7-10　ドライバ・コンフィグレーション(RL78/F23,F24 スレーブ) (2/2) 94
表 7-11　ドライバ・コンフィグレーション(RL78/F23,F24 マスター) (1/2) 96
表 7-12　ドライバ・コンフィグレーション(RL78/F23, F24 マスター) (2/2) 98
表 8-1　LIN2.1 ソフトウェア・スレーブ・ドライバ関数一覧 99
表 8-2　LIN2.1spec と LIN2.1 スレーブ・ドライバ型定義の対応一覧 100
表 8-3　LIN2.1 ソフトウェア・ドライバとクラスタ管理関数一覧（スレーブ） 103
表 8-4　スカラ・シグナル読み出し関数一覧（スレーブ） 105
表 8-5　スカラ・シグナル書き込み関数一覧（スレーブ） 111
表 8-6　バイト・アレイ読み出し関数一覧（スレーブ） 116
表 8-7　バイト・アレイ書き込み関数一覧（スレーブ） 119
表 8-8　通知関数一覧（スレーブ） 122
表 8-9　インタフェース・マネージメント関数一覧（スレーブ） 125
表 8-10　ユーザ定義コール・アウト関数一覧（スレーブ） 131

表 9-1　LIN2.1 ソフトウェア・マスター・ドライバ関数一覧 136
表 9-2　LIN2.1spec と LIN2.1 マスター・ドライバ型定義の対応一覧 137
表 9-3　LIN2.1 ソフトウェア・ドライバとクラスタ管理関数一覧（マスター） 140
表 9-4　スカラ・シグナル読み出し関数一覧（マスター） 142
表 9-5　スカラ・シグナル書き込み関数一覧（マスター） 148
表 9-6　バイト・アレイ読み出し一覧（マスター） 153
表 9-7　バイト・アレイ書き込み関数一覧（マスター） 156
表 9-8　通知関数一覧（マスター） 159
表 9-9　スケジュール・マネージメント関数一覧（マスター） 162
表 9-10　インタフェース・マネージメント関数一覧（マスター） 167
表 9-11　ノード・コンフィグレーション関数一覧（マスター） 175
表 9-12　error_code 一覧 178
表 9-13　PID リスト設定値 180
表 9-14　ユーザ定義コール・アウト関数一覧（マスター） 184

図　目　次

図 3-1　システム概要 20
図 3-2　LIN システム構築手順 21
図 3-3　ユーザ･アプリケーションと LIN2.1 ソフトウェア・ドライバの関係 23
図 4-1　スマート・コンフィグレータとの連携イメージ 25
図 5-1　LIN アプリケーションビルドフロー 30
図 5-2　device.h 内インクルード記載方法 35
図 5-3　LIN アプリケーションの構築方法 36
図 5-4　RL78/F23,F24 のスレーブ割り込み処理タイミングイメージ 39
図 5-5　conflin_x.h 内マクロ記載方法 40
図 6-1　メイン画面 49
図 6-2　メイン画面（デバイス選択前） 50
図 6-3　デバイス設定ウィンドウ（デバイス選択前） 53
図 6-4　デバイス設定ウィンドウ（複数チャネル選択可能デバイス） 53
図 6-5　メイン画面（チャネル設定前） 54
図 6-6　チャネル設定ウィンドウ 55
図 6-7　メイン画面（チャネル項目設定前） 56
図 6-8　ボー・レート設定ウィンドウ 57
図 6-9　メッセージ管理ウィンドウ 58
図 6-10　無条件フレーム設定ウィンドウ 59
図 6-11　フレーム内信号管理ウィンドウ 60
図 6-12　信号設定ウィンドウ 61
図 6-13　イベント・トリガ・フレーム設定ウィンドウ 62
図 6-14　スポラディック・フレーム設定ウィンドウ 63
図 6-15　スケジュール・テーブル管理ウィンドウ 64
図 6-16　スケジュール・テーブル設定ウィンドウ 65
図 6-17　スケジュール・テーブル（Entry#1、#2、#3）の動き 65
図 6-18　Setting Node ウィンドウ 66
図 6-19　Setting Others ウィンドウ 67
図 6-20　ソース・コード出力先 68
図 7-1　メッセージ・データへのシグナル配置例 73
図 7-2　フレーム構成 74
図 7-3　バイト・フィールド・フォーマット 75
図 7-4　Break フォーマット 75
図 7-5　データ送受信例 77
図 7-6　イベント･トリガ･フレームによるデータ送受信例 78
図 7-7　スポラディック･フレームによるデータ送受信例 81
図 7-8　l_sch_tick 関数とフレーム転送の関係 89
図 7-9　l_sch_set 関数によるスケジュール・テーブルの切り替え 90
図 8-1　LIN2.1 ソフトウェア・スレーブ・ドライバ関数の記述フォーマット 101
図 9-1　LIN2.1 ソフトウェア・マスター・ドライバ関数の記述フォーマット 138

第1章 製品概要

1. 1 概要

LIN2.1 ソフトウェア・ドライバは、ルネサスエレクトロニクス製のマイクロコンピュータで LIN 通信を実現するためのソフトウェア・ドライバです。LIN 2.1Spec に準拠したアプリケーション・プログラム・インタフェース関数（API 関数）を提供しています。

本 LIN 2.1 ソフトウェア・ドライバは以下のマイクロコンピュータに対応しています。

・16 ビット・マイクロコンピュータ RL78/F23
・16 ビット・マイクロコンピュータ RL78/F24

1. 2 特徴

1. 2. 1 高い移植性

ユーザがLINのハードウェア依存箇所、CPUコアを意識せずにLIN通信プログラムを記述することを可能にしています。これにより、異なる実行環境への移植を容易にしています。

1. 2. 2 コンフィギュレーション・ツールの提供

ユーザの使用するデバイス、環境に則したLINのハードウェア初期設定、メッセージなどの静的生成がGUIを中心とした操作により容易に設定可能です。

1. 3　LIN2.1 ソフトウェア・ドライバの種類

LIN2.1 ソフトウェア･ドライバには、以下の種類が存在します。

表 1-1　ノード情報一覧

種別	説明
ハードウェア依存	デバイス・シリーズごとに用意されています。
マスター/スレーブ機能	マスター・アプリケーション用とスレーブ･アプリケーション用の2種類が用意されています。

1. 4　実行環境

LIN2.1 ソフトウェア・ドライバは、次のハードウェアを備えたターゲット・システム上で動作します。

1. 4. 1　対象マイコン

（1）スレーブ

表 1-2　対象マイコン一覧（スレーブ）

シリーズ名	品種
RL78シリーズ	RL78/F23、RL78/F24

（2）マスター

表 1-3　対象マイコン一覧（マスター）

シリーズ名	品種
RL78シリーズ	RL78/F23、RL78/F24

1. 4. 2 メモリ容量

メモリ容量は、アプリケーションにて使用する関数の種類、メッセージ数と利用する周辺I/O、コンパイラ、コンパイル条件等により変化します。全関数を利用した場合のメモリ容量は次の通りです。

（1）スレーブ

表 1-4 メモリ容量（スレーブ）

部品名	ROM 容量	RAM 容量
LIN2.1 ソフトウェア・ドライバ本体	3.5K バイト（RL78/F23,F24：CC-RL） 3.6K バイト（RL78/F23,F24：IAR）	28 バイト（RL78/F23,F24：CC-RL） 25 バイト（RL78/F23,F24：IAR）
スタック	-	約 200 バイト
1 無条件フレーム	5 バイト	14 バイト（16 信号版） 18 バイト（32 信号版）
1 信号	4 バイト（16 信号版） 5 バイト（32 信号版）	-
1 イベント・トリガ・フレーム (無条件フレームとの関連毎に)	2 バイト	1 バイト

注：上記以外に、送受信メッセージ用に確保される容量が必要となります。

注：オート・ボー・レート・モードかつ、ウェイクアップ方法として立ち下がりエッジを選択した場合の値になります。

注：フレーム、信号の使用メモリ量は構造体メンバのアラインを考慮しない値です。

注：16 信号版はコンパイラ・オプション __LIN_SIGNAL_32__を使用していないドライバ、32 信号版は__LIN_SIGNAL_32__を使用しているドライバを意味しています。

（2）マスター

表 1-5 メモリ容量（マスター）

部品名	ROM 容量	RAM 容量
LIN2.1 ソフトウェア・ドライバ本体	4.8K バイト（RL78/F23,F24：CC-RL） 4.6K バイト（RL78/F23,F24：IAR）	46 バイト（RL78/F23,F24：CC-RL） 45 バイト（RL78/F23,F24：IAR）
スタック	-	約 150 バイト
1 無条件フレーム	12 バイト	11 バイト(16 信号版) 13 バイト(32 信号版)
1 信号	5 バイト	-
1 イベント・トリガ・フレーム	17 バイト	-
1 スポラディック・フレーム	16 バイト	-
1 スケジュール・エントリ	4 バイト	-

注：上記以外に、送受信メッセージ用に確保される容量が必要となります。

注：フレーム、信号、スケジュールの使用メモリ量は構造体メンバのアラインを考慮しない値です。

注：16 信号版はコンパイラ・オプション __LIN_SIGNAL_32__を使用していないドライバ、32 信号版は__LIN_SIGNAL_32__を使用しているドライバを意味しています。

1. 4. 3　利用資源

各利用資源を以下に示します。

（1）スレーブ

シリアル・インタフェース：RLIN3 チャネル 0、RLIN3 チャネル 1

タイマ･アレイ･ユニット：TAU00～07、TAUA10～17

外部割込み：INTP11 (INTLIN0WUP)、INTP12 (INTLIN1WUP) (※)

※外部割込みは、LIN ドライバのウェイクアップ方式を立ち下がりエッジ検出方式に設定している場合のみ使用します。ウェイクアップ方式がドミナント幅検出方式の場合は使用しません。

表 1-6　利用資源一覧(スレーブ)

マイコン	UART※1		TAU※2	
	チャネル	割り込み要求	ユニット/チャネル	割り込み要求
RL78/F23	RLIN3(ﾁｬﾈﾙ 0)	送信完了割り込み:INTLIN0TRM 受信完了割り込み:INTLIN0RVC エラー割り込み:INTLIN0STA 外部割込み:INTLIN0WUP	TAU00~07 TAU10~13	INTTM00～07 INTTM10～13
RL78/F24	RLIN3(ﾁｬﾈﾙ 0)	送信完了割り込み:INTLIN0TRM 受信完了割り込み:INTLIN0RVC エラー割り込み:INTLIN0STA 外部割込み:INTLIN0WUP	TAU00~07 TAU10~17	INTTM00～07 INTTM10～17
	RLIN3(ﾁｬﾈﾙ 1)	送信完了割り込み:INTLIN1TRM 受信完了割り込み:INTLIN1RVC エラー割り込み:INTLIN1STA 外部割込み:INTLIN1WUP		

※1 : UART チャネルは、いずれか 1 チャネルのみ選択可能です。

※2 : TAU ユニット/チャネルは、いずれか 1 ユニット/チャネルのみ選択可能です。

（2）マスター

シリアル・インタフェース：RLIN3 チャネル 0、RLIN3 チャネル 1

外部割込み：INTP11 (INTLIN0WUP)、INTP12 (INTLIN1WUP)　(※)

※外部割込みは、LIN ドライバのウェイクアップ方式を立ち下がりエッジ検出方式に設定している場合のみ使用します。ウェイクアップ方式がドミナント幅検出方式の場合は使用しません。

表 1-7　利用資源一覧(マスター)

マイコン	UART※1	
	チャネル	割り込み要求
RL78/F23	RLIN3(ﾁｬﾈﾙ 0)	送信完了割り込み:INTLIN0TRM 受信完了割り込み:INTLIN0RVC エラー割り込み:INTLIN0STA 外部割込み:INTLIN0WUP
RL78/F24	RLIN3(ﾁｬﾈﾙ 0)	送信完了割り込み:INTLIN0TRM 受信完了割り込み:INTLIN0RVC エラー割り込み:INTLIN0STA 外部割込み:INTLIN0WUP
	RLIN3(ﾁｬﾈﾙ 1)	送信完了割り込み:INTLIN1TRM 受信完了割り込み:INTLIN1RVC エラー割り込み:INTLIN1STA 外部割込み:INTLIN1WUP

※1：UART チャネルは、いずれか 1 チャネルのみ選択可能です。

1. 5　開発環境

LIN2.1 ソフトウェア・ドライバを使用したアプリケーション・システムを開発する上で必要となる環境を次に表示します。

（１）ハードウェア

ホストマシン　　：IBM-PC/AT™ 互換機
OS　　：Windows 10
(Microsoft .NET Framework 3.5 が必要です)
※上記はご使用のコンパイラ及び、ディバッガの使用条件に従います。

（２）ソフトウェア

表 1-8　ソフトウェア一覧

マイコン	統合開発環境	コンパイラ	ディバッガ
RL78/F23,F24	CS+	CC-RL	CS+ 付属
	IAR	IAR compiler	IAR compiler 付属

1. 6　制約事項

本ソフトウェアを使用する上での制約事項を以下に示します。

1. 6. 1　クロック、ボー・レート設定

CPU／周辺ハードウェア・クロック（f_{CLK}）をLIN通信クロック源に使用する場合、f_{CLK}及びボー・レートは以下の条件で使用してください。

表 1-9　使用条件一覧

種別	使用条件	
	周辺H/Wクロック (fCLK)	ボー・レート(BR)
マスター・ドライバ	4 MHz ≦ fCLK ≦ 40 MHz	9600 bps, 19200 bps
スレーブ・ドライバ （オート・ボー・レート）	8 MHz ≦ fCLK ≦ 40 MHz	2400 bps ≦ BR ≦ 20000 bps
スレーブ・ドライバ （固定ボー・レート）	4 MHz ≦ fCLK ≦ 40 MHz	9600 bps, 19200 bps

高速システム・クロック（f_{MX}）をLIN通信クロック源に使用する場合は、f_{MX}とf_{CLK}を以下の条件で使用してください。ボー・レートの範囲は上記の表と同じです。

4 MHz ≦ f_{MX} ≦ 40 MHz かつ f_{CLK} ≧ f_{MX} x 1.2

（ただし、スレーブのオート・ボー・レート使用時は4 MHz ≦ f_{MX} ＜ 8 MHz不可）

スマート・コンフィグレータおよびLINコンフィギュレータで指定可能なLIN通信クロック源周波数は以下のとおりです。これ以外の周波数を使用する場合は、出力ファイルを別途編集する必要があります。

8, 10, 12, 16, 20, 24, 32, 40 [MHz]

1. 6. 2　割り込みに関する制約事項

ドライバが使用している割り込みと同じリソースを共有している割り込みをユーザ・アプリケーション側で同時に使用した場合、ドライバは正しく動作しません。ドライバが使用する割り込みは 1. 4. 3 利用資源を参照してください。

割り込みベクタ・テーブルの仕様については、RL78/F23,F24 ユーザーズ・マニュアル（割り込み要因一覧）を参照してください。

1. 6. 3 その他の制約事項

- RL78/F23,F24 ドライバでは、API や割り込み処理内で LIN チャネル選択レジスタ LCHSEL の値を書き換えますが、書き換え前の値に戻さないので注意してください。

- RL78/F23,F24 スレーブ・ドライバが使用するタイマ・アレイ・ユニットのチャネルと同じチャネルをアプリケーションで使用しないでください。また、スレーブ・ドライバが使用するタイマ・アレイ・ユニットのクロック選択と同じクロック選択をアプリケーションで使用しないでください。

- 本ドライバの評価において、LIN 仕様を逸脱する設定(ドライバ・コンフィグレーションにおいて”ビット・エラー検出を禁止する”など)の項目は評価を行っておりません。LIN 仕様外の設定を使用する場合は、お客様の環境にて動作確認を行ってください。

- RL78/F23,F24 マスター・ドライバでは、自ハードウェアに起因するエラーが発生した場合にコールバック関数 l_sys_call_fatal_error によってユーザに通知を行います。詳細は 9. 3. 10 を参照してください。

- RL78/F23,F24 マスター・ドライバでは、conf/conflin_x.c 内のマクロ定義 CONFLIN_u2sBAUDRATEWKUP によってドライバ・スリープ中のボー・レートを変更可能ですが、通常は 19200[bps]から変更しないでください。ウェイクアップ・リクエストの受信が正しく行えなくなる可能性があります。

- RL78F23,F24 ドライバでは、ドライバ・コンフィグレーション CONFMLIN_OPT_u1gLINMCK_CFG によって LIN 通信クロック源の選択を行います。LIN コンフィギュレータで出力された後、適切な選択を行ってください。

第2章 インストレーション

2. 1 概要

LIN2.1 ソフトウェア・ドライバ関数は、コンフィギュレータで提供されます。

2. 2 インストール

2. 2. 1 LIN コンフィギュレータのインストール

インストーラ LINConfigurator_RL78F23_F24_J_V100.msiを実行します。
スマート・コンフィグレータの出力では、r_lin3/tool フォルダに上記インストーラが格納されます。

【注意事項】
LINコンフィギュレータを使用する際は、.NET Framework 3.5が必要です。
.NET Framework 3.5は、マイクロソフト社のWebより取得できます。

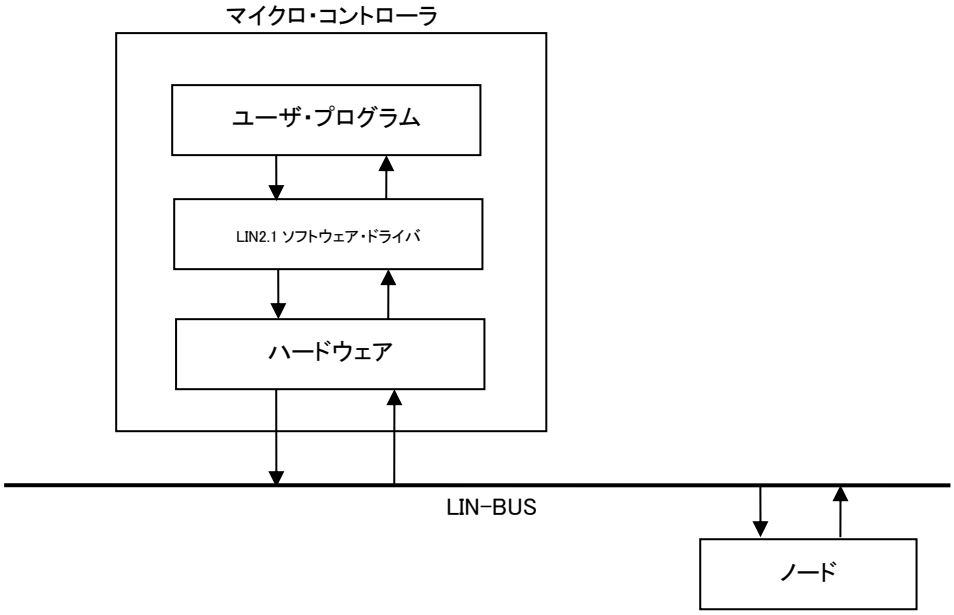
第3章 システム構築概要

3. 1 LIN2.1 ソフトウェア・ドライバの位置づけ

LIN2.1 ソフトウェア・ドライバは、システム内においてユーザ・アプリケーションとハードウェアの間に位置しています（図 3-1 参照）。また、ハードウェア制御を行うためのユーザ･インタフェースを持っています。

ユーザは、アプリケーション内に LIN2.1 ソフトウェア・ドライバ関数を記述することにより、ハードウェアのレジスタ制御を意識する必要がなくなります。

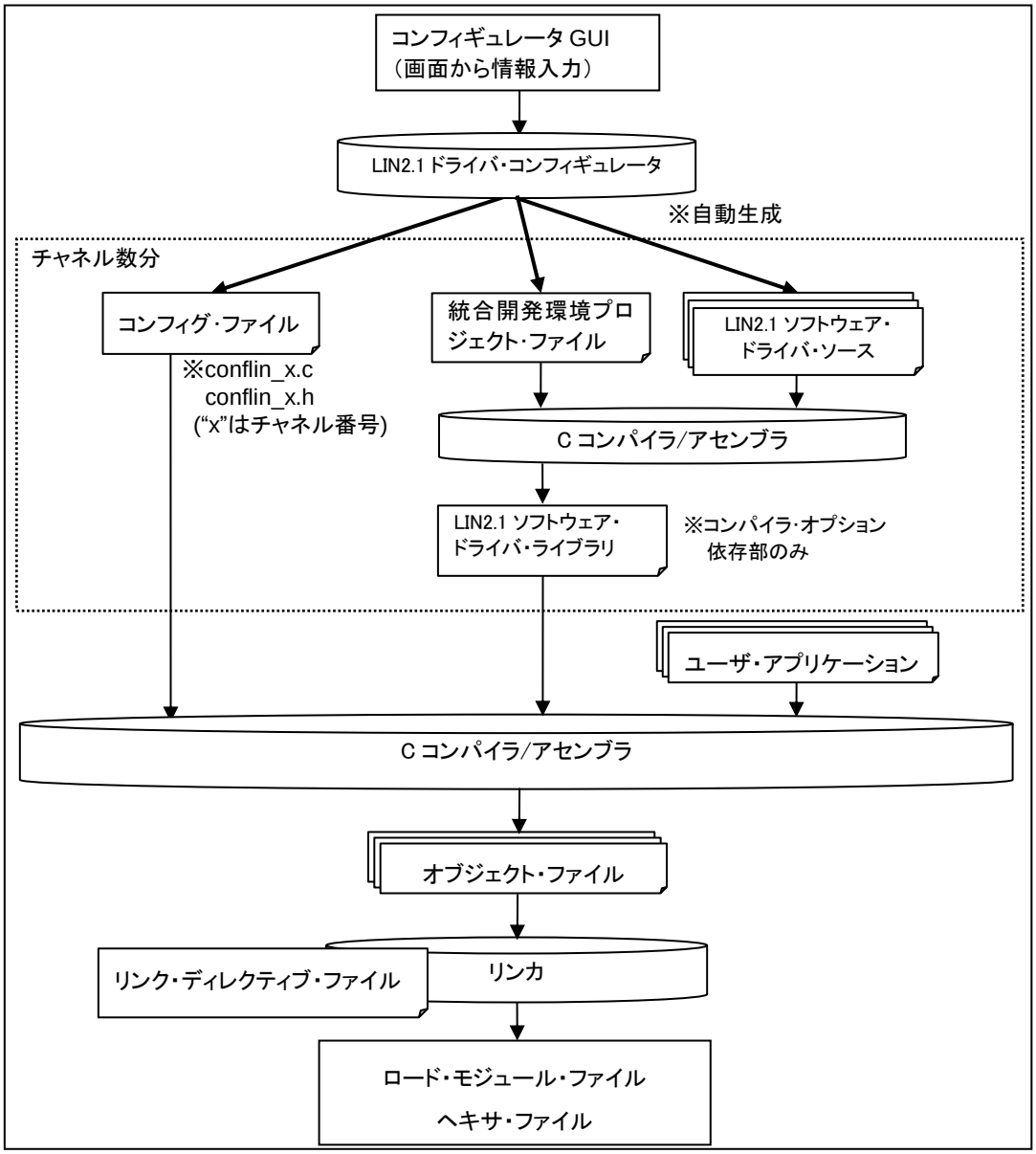
図3-1 システム概要



3. 2 システム構築手順

コンフィギュレータ、LIN2.1 ソフトウェア・ドライバを用いてシステム構築する手順を示します。
（conflin_x.c、conflin_x.h(“x”はチャネル番号)を直接編集することも可能です。）

図3-2 LINシステム構築手順



備考：LIN2.1 ソフトウェア・ドライバ・ライブラリには、コンパイラ・オプションに依存するソース・ファイルが含まれます。ライブラリのコンパイラ・オプションについては「5. 1. 1 ライブラリ用コンパイラ・オプション」を参照してください。

3. 2. 1　LIN コンフィギュレータによるファイル生成

LINコンフィギュレータにより、使用デバイス、周辺ハードウェア・クロック、LINのボー・レート、各種フレーム、スケジュール、ノード情報を設定し、情報ファイル、ヘッダ・ファイルの形で生成します。

LINコンフィギュレータによる設定手順の詳細は「第6章 LINコンフィギュレータ」を参照してください。

（１） LINコンフィギュレータの入力データ

LINコンフィギュレータによるファイル生成を行う前には、最低限、次に示す内容を決定しておく必要があります。

・使用するデバイス（シリーズ名称、デバイス名）
・デバイスの周辺ハードウェア・クロック
・使用するチャネル
・LINボー・レート
・シグナル設定
・フレーム設定
・スケジュール設定
・ノード情報

（２） LINコンフィギュレータの出力データ

LINコンフィギュレータからは以下のファイルが出力されます。

･コンフィグ･ファイル
・統合開発環境プロジェクト・ファイル
・LIN2.1ソフトウェア・ドライバ・ソース・ファイル（コンパイラ・オプション依存部）

（３） コンパイラ･オプション依存部ライブラリの生成

LINコンフィギュレータからファイルを生成した後、コンパイラ･オプション依存部のライブラリを生成してください。

統合開発環境のプロジェクト・ファイルを用いることで、任意のコンパイラ･オプションを指定してLIN2.1ソフトウェア・ドライバ・ソース・ファイルから、コンパイラ･オプション依存部のライブラリを生成することができます。

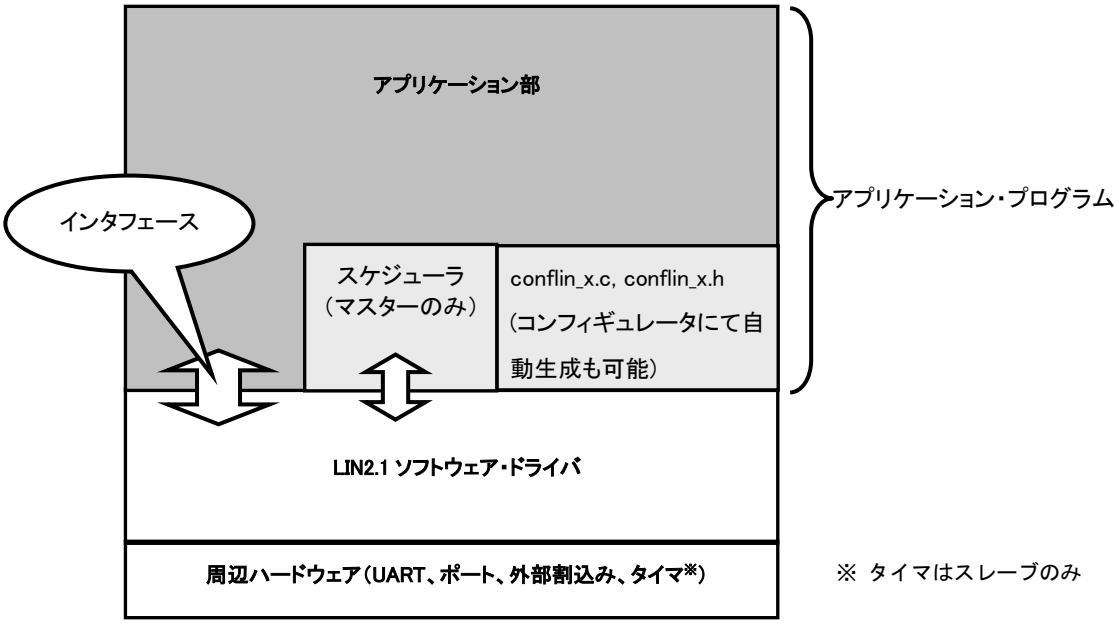
LINシステムの構築には、コンパイラ･オプション依存部、ライブラリが必要となります。

3. 2. 2 ユーザ・アプリケーション

LIN2.1ソフトウェア･ドライバ関数を使用してLIN通信アプリケーションを作成します。ユーザ・アプリケーションは、アプリケーション部と、conflin_x.c、conflin_x.h(“x”はチャネル番号)から構成されます。

LIN2.1ソフトウェア・ドライバ関数を使用するファイルには、コンフィギュレータで生成されたヘッダ･ファイルのインクルードが必要です。

図3-3 ユーザ･アプリケーションとLIN2.1ソフトウェア・ドライバの関係



詳細は「第5章 LINアプリケーションビルド方法」を参照してください。

3. 2. 3　ビルド

（1）オブジェクト・ファイルの生成

ユーザ・アプリケーションとLINコンフィギュレータから生成されたファイル群を用いて、コンパイル/アセンブルを実行し、リロケータブルなオブジェクト・ファイルを生成します。

備考　Cコンパイラ/アセンブラの起動オプション、及び実行方法についての詳細は、各ツールのユーザーズ・マニュアルを参照してください。

（2）ロード・モジュール・ファイルの生成

次のファイル群に対して、リンクを実行してロード・モジュール・ファイルを生成します。

・ユーザ・アプリケーションをコンパイル/アセンブルしたオブジェクト・ファイル
・LINコンフィギュレータで作成された情報ファイルをコンパイルしたオブジェクト・ファイル
（コンパイラ･オプション依存部のライブラリ・ファイル）
・リンク・ディレクティブ・ファイル
・Cコンパイラ・パッケージが推奨するライブラリ・ファイル

第4章 スマート・コンフィグレータとの連携

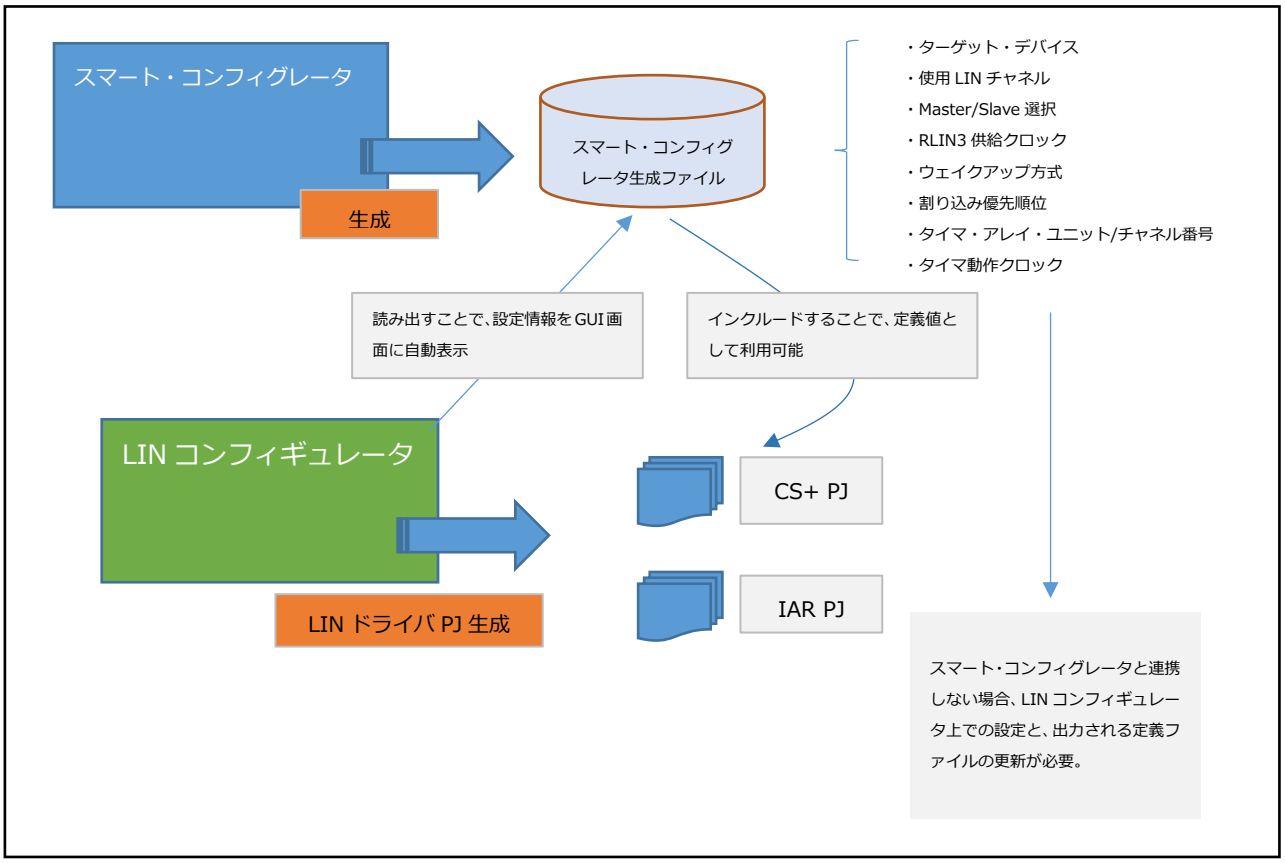
スマート・コンフィグレータは、「ソフトウェアを自由に組み合わせられる」をコンセプトとしたユーティリティです。ミドルウェアとドライバをインポート、端子設定で、お客様のシステムへのルネサス製ドライバの組み込みを容易にします。

LINコンフィギュレータは、特定のデバイス向けとして、スマート・コンフィグレータによって生成されたLIN関連の設定情報が記述されたヘッダ・ファイル（r_rlin3_config.h）を読み出す機能がサポートされています。

該当ファイルを読み出すことで、スマート・コンフィグレータ上で設定されたLIN関連の設定情報が、LINコンフィギュレータ上の画面に自動的に反映されます。

また、LINコンフィギュレータから生成されるLIN 2.1ドライバに該当ファイルを組み込むことで、conflin_x.hファイルの一部定義値が、スマート・コンフィグレータ上で設定された定義値に置き換わります。

図 4-1　スマート・コンフィグレータとの連携イメージ



4.1 操作手順

以下に、スマート・コンフィグレータとLINコンフィギュレータとの連携に関する一連の操作手順を示します。
スマート・コンフィグレータの操作についての詳細は、スマート・コンフィグレータのユーザーズ・マニュアルを参照してください。

① スマート・コンフィグレータ上でLIN関連（RLIN3）のパラメータを設定し、コード生成を行います。
生成されるファイルは、以下のとおりです。

・r_rlin3_config.h : LINコンフィグ値を設定したファイル
・r_rlin3_m_callout.c, r_rlin3_s_callout.c : ユーザ定義コール・アウト関数
・conflin_x.c , conflin_x.h : 空ファイル（LINコンフィギュレータの出力ファイルと置き換えるファイル）

r_rlin3_config.hファイルについては後述します。

② LINコンフィギュレータから、①で生成したr_rlin3_config.hファイルを読み出します。
6. 2. 1 LINコンフィギュレータの起動画面のFileメニューから“SMC Header File Open”を選択します。

③ ファイルの読み出しに成功した場合、スマート・コンフィグレータ上で設定された情報が6. 2. 3 デバイス選択画面と6. 2. 4 チャネル設定画面に自動的に反映されます。
反映される項目は次のとおりです。

[6. 2. 3 デバイス選択画面]
Series Information
Device Name
Channel

[6. 2. 4 チャネル設定画面]
Master/Slave
Peripheral Hardware Clock

④ LINコンフィギュレータ上で他の項目の設定を行い、コード生成を行います。

⑤ 生成されたフォルダ中に含まれるr_rlin3_config.hファイルを、①で生成したファイルと置き換えます。
これにより、confmlin_opt.h, confslin_opt.hファイルの一部定義値が、r_rlin3_config.hファイル中の定義値を参照することになります。

4. 2 コンフィグ値

r_rlin3_config.hファイルは、LINコンフィギュレータが読み取ってGUIを適切に設定するための定義と、LINドライバが参照するための定義とで構成されます。RLIN3やタイマのチャネルは、スマート・コンフィグレータ上から、対象のデバイスに搭載されているチャネル数の範囲内で設定してください。

4. 2. 1 LIN コンフィギュレータ用設定

以下の情報は、LINコンフィギュレータが読み取ってGUIに表示します。

表 4-1 LIN コンフィギュレータ用定義値 (r_rlin3_config.h)

定義式	意味
RLIN3_CFG_DEVICENAME	ターゲット・デバイス名
RLIN3_CFG_USE_LIN_CH0 RLIN3_CFG_USE_LIN_CH1	LINチャネル0、1ごとに使用するLIN端子設定 LINチャネル0 0: 未使用 1: P13,P14 2: P42,P43 LINチャネル1 0: 未使用 1: P10, P11 2: P106,P107 3: P120,P125
RLIN3_CFG_CH0_OPERATION_MODE RLIN3_CFG_CH1_OPERATION_MODE	LINチャネル0、1ごとのMaster/Slaveの選択 0: マスター 1: スレーブ
RLIN3_CFG_CH0_INPUT_CLOCK RLIN3_CFG_CH1_INPUT_CLOCK	LINチャネル0、1への供給クロック (*1) 0: 40 MHz 1: 32 MHz 2: 24 MHz 3: 20 MHz 4: 16 MHz 5: 12 MHz 6: 10 MHz 7: 8 MHz

【注意事項】

1. スマート・コンフィグレータ上で不整合となる設定は回避してください。

4. 2. 2 LIN ドライバ用設定

以下の情報は、LIN コンフィギュレータより生成される LIN ドライバが参照します。

LIN ドライバ・ライブラリのビルドの際、r_rlin3_config.h をインクルードするよう設定してください。

表 4-2 LIN ドライバ用定義値 (r_rlin3_config.h)

定義式	意味
RLIN3_CFG_CH0_BUSWAKEUP RLIN3_CFG_CH1_BUSWAKEUP	LINチャネル0、1のウェイクアップ方式 0x00: 立ち下がりエッジ検出方式 0x01: ドミナント幅検出方式
RLIN3_CFG_INTLIN0TRM_PRIORITY_LEVEL RLIN3_CFG_INTLIN1TRM_PRIORITY_LEVEL	INTLIN0TRM、INTLIN1TRM割り込み優先順位 0: Level 0 ※最高優先度 1: Level 1 2: Level 2 3: Level 3
RLIN3_CFG_INTLIN0RVC_PRIORITY_LEVEL RLIN3_CFG_INTLIN1RVC_PRIORITY_LEVEL	INTLIN0RVC、INTLIN1RVC割り込み優先順位 0: Level 0 ※最高優先度 1: Level 1 2: Level 2 3: Level 3
RLIN3_CFG_INTLIN0STA_PRIORITY_LEVEL RLIN3_CFG_INTLIN1STA_PRIORITY_LEVEL	INTLIN0STA、INTLIN1STA割り込み優先順位 0: Level 0 ※最高優先度 1: Level 1 2: Level 2 3: Level 3
RLIN3_CFG_INTLIN0WUP_PRIORITY_LEVEL RLIN3_CFG_INTLIN1WUP_PRIORITY_LEVEL	INTLIN0WUP、INTLIN1WUP割り込み優先順位 0: Level 0 ※最高優先度 1: Level 1 2: Level 2 3: Level 3
RLIN3_CFG_CH0_TAU_UNIT RLIN3_CFG_CH1_TAU_UNIT	LIN Slaveドライバが使用するタイマ・アレイ・ユニット番号 0x00: ユニット0 0x01: ユニット1
RLIN3_CFG_CH0_TAU_CH RLIN3_CFG_CH1_TAU_CH	LIN Slaveドライバが使用するタイマ・アレイ・ユニットのチャネル番号 0x00: チャネル0 0x01: チャネル1 0x02: チャネル2 0x03: チャネル3 0x04: チャネル4 0x05: チャネル5

	0x06: チャネル6 0x07: チャネル7
RLIN3_CFG_CH0_TAU_CLKSEL RLIN3_CFG_CH1_TAU_CLKSEL	LIN Slaveドライバが使用するタイマ・アレイ・ユニットの動作クロック 0x0000: タイマ・クロック選択レジスタm（TPSm）で設定した動作クロックCKm0 0x8000: タイマ・クロック選択レジスタm（TPSm）で設定した動作クロックCKm1
RLIN3_CFG_CH0_TAU_INTTMmn_PRIORITY_LEVEL RLIN3_CFG_CH1_TAU_INTTMmn_PRIORITY_LEVEL	LIN Slaveドライバが使用するタイマ・アレイ・ユニットのINTTMmn割り込み優先順位 0: Level 0 ※最高優先度 1: Level 1 2: Level 2 3: Level 3
RLIN3_CFG_USE_SMC_DEFINITION	conflin_x.hファイルの一部定義値は上記の定義値となる。

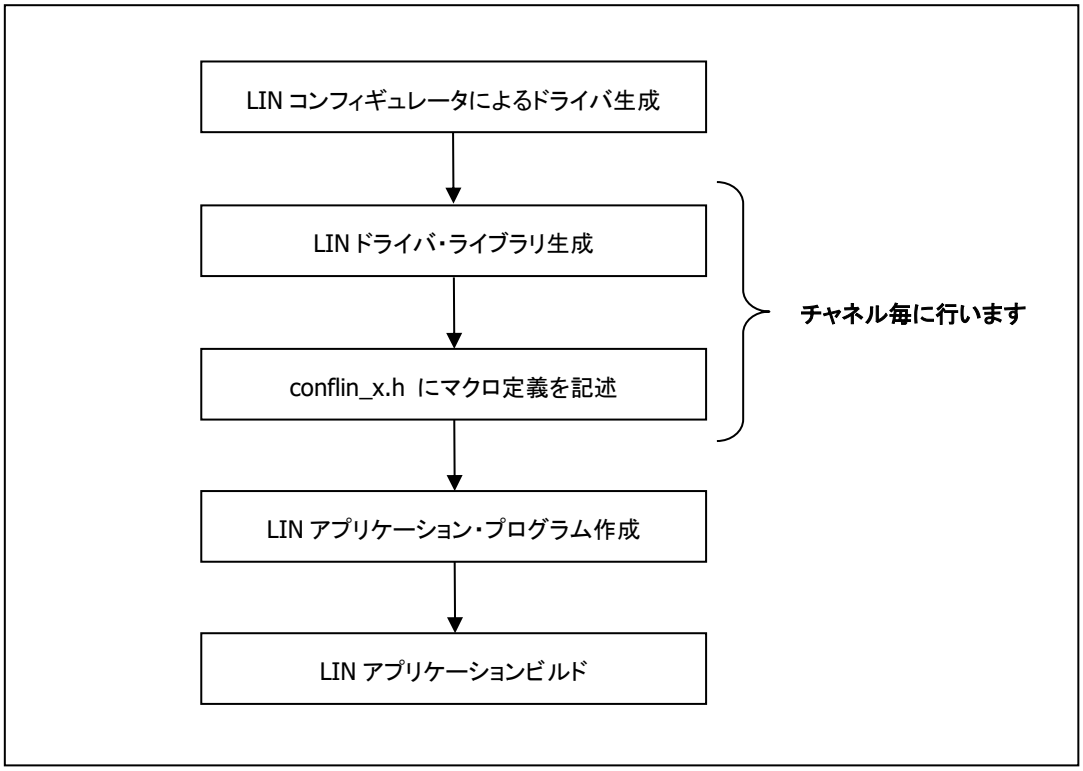
【注意事項】

1. **スマート・コンフィグレータ上で設定する LIN 関連の設定が、他の機能の設定と競合しないようにしてください。注意が必要となるのは次の機能設定となります。**
   - **INTLIN0TRM、INTLIN0RVC、INTLIN0STA、INTLIN0WUP、INTLIN1TRM、INTLIN1RVC、INTLIN1STA、INTLIN1WUP の各割り込み優先順位**
   - **タイマ・アレイ・ユニットの使用ユニット／チャネル番号、および割り込み優先順位**
2. **製品が搭載するユニット数やチャネル数の範囲内で設定してください。**

第5章 LIN アプリケーションビルド方法

LIN コンフィギュレータによって出力されたソース・コードをビルドし、アプリケーションに組み込む方法を説明します。

図5-1　LINアプリケーションビルドフロー



LIN アプリケーションの作成(ビルド)のフローは上記の通りです。

「LIN コンフィギュレータによるドライバ生成」に関しては「第 6 章 LIN コンフィギュレータ」を参照してください。

5.1 LIN ドライバ・ライブラリ生成

LIN コンフィギュレータによって出力された LIN ドライバ・ソースコードをコンパイル、リンクし、LIN ドライバ・ライブラリを生成します。LIN ドライバ・ライブラリはチャネル毎にビルドする必要があります。

LIN コンフィギュレータが出力した開発環境プロジェクト・ファイルを開発環境で読み込み、ビルドすると LIN ドライバのライブラリ・ファイルが生成されます。

CS+ ：マスタ liblin21m_CCRL_x.lib、スレーブ liblin21s_CCRL_x.lib

IAR ：マスタ liblin21m_IAR_x.a、スレーブ liblin21s_IAR_x.a

x = 0, 1 (チャネル番号)

ライブラリをビルドする前に、必要に応じて以下の編集が必要です。

1) コンパイル・オプション

LIN コンフィギュレータによって設定できない項目をコンパイル・オプションとして指定し、ドライバの動作を変更することが可能です。詳細は「5. 1. 1 ライブラリ用コンパイラ・オプション」と LIN コンフィギュレータによって出力される readme.txt を参照してください。

2) r_rlin3_config.h 指定（スマート・コンフィグレータ連携時）

LIN コンフィギュレータはスマート・コンフィグレータと連携しない場合を想定し、空の r_rlin3_config.h をフォルダ libsrc/conf/に出力します。スマート・コンフィグレータと連携する場合は、空の r_rlin3_config.h をスマート・コンフィグレータの出力ファイルで上書きしてください。

3) ドライバ・コンフィグレーション

LIN ドライバではドライバ・コンフィグレーションによってドライバの性質を変更することが可能です。必要に応じてドライバ・コンフィグレーション(スレーブは libsrc/conf/confslin_opt.h、マスターは libsrc/conf/confmlin_opt.h)を修正してからビルドを行ってください。詳細は「5. 1. 2 confmlin_opt.h 編集（マスター用）」「5. 1. 3 confslin_opt.h 編集（スレーブ用）」を参照してください。

4) I/O ヘッダ・ファイル指定（IAR のみ）

LIN ドライバはSFRレジスタにアクセスするため、レジスタのアドレス定義が記載されたI/Oヘッダ・ファイルを必要とします。使用する開発環境がCS+の場合はビルド時に自動的に解決されますが、IARの場合は別途編集が必要です。詳細は「5. 1. 4 IAR I/Oヘッダ・ファイル指定」を参照してください。

5. 1. 1　ライブラリ用コンパイラ・オプション

RL78/F23,F24ドライバ・ライブラリ生成時のコンパイラ・オプションを以下に示します。

LINコンフィギュレータが出力する開発環境プロジェクト・ファイルでは、「必須」「基本的に設定」とするオプションを定義済みです。

チャネルのコンパイラ･オプションの初期設定は__LIN_CH0_P1__または__LIN_CH1_P1__が定義されます。スマート・コンフィグレータと連携しないとき、ポート1以外を使用する場合はプロジェクト・ファイルを編集してオプションを変更してください。スマート・コンフィグレータを使用する場合は、指定したポートに合わせて自動的にオプションの置き換えが行われるため編集不要です。

「必要に応じて選択」のオプションが必要な場合、または「基本的に設定」のオプションを使用しない場合は、プロジェクト・ファイルの編集を行ってください。編集方法は、使用する統合開発環境のユーザーズ・マニュアルを参照してください。

マクロ__LIN_SIGNAL_32__を使用する場合は、conflin_x.hにも別途追加が必要です。詳細は「5. 2. 4 コンパイラ・オプション編集（conflin_x.h）」を参照してください。

表 5-1　ライブラリ用コンパイラ・オプション一覧

コンパイラ・オプション	マクロ名	RL78/F23,F24 マスター	RL78/F23,F24 スレーブ
対象マイコン	__LIN_RL78_F23_F24__	◎	◎
チャネル	※1	◎	◎
メモリコピールーチンのアセンブラ版	__LIN_MEMCOPY_ASM__	△※2	△※2
HWによる自動ボー・レート検出あり	__LIN_HW_AUTO_BR__	対象外	○※3 ※4
フレーム内信号数32	__LIN_SIGNAL_32__	△	△

◎：必須、 ○：基本的に設定、 △：必要に応じて選択

※1　__LIN_CH0_P1__, __LIN_CH0_P4__, __LIN_CH1_P1__, __LIN_CH1_P10__, __LIN_CH1_P12__ のいずれか。
スマート・コンフィグレータと連携しない場合は、使用するポートグループ番号に合わせてコンパイラ・オプションを変更してください。
・デバイスとしてチャネル0を指定する場合
　ポート1使用の場合　：__LIN_CH0_P1__　（受信端子P14、送信端子P13）
　ポート4使用の場合　：__LIN_CH0_P4__　（受信端子P43、送信端子P42）
・デバイスとしてチャネル1を指定する場合
　ポート1使用の場合　：__LIN_CH1_P1__　（受信端子P11、送信端子P10）
　ポート10使用の場合　：__LIN_CH1_P10__　（受信端子P107、送信端子P106）
　ポート12使用の場合　：__LIN_CH1_P12__　（受信端子P125、送信端子P120）

※2　IARコンパイラにおいては、メモリ・モデルをnearモデルに指定した場合のみ使用可能です。
CC-RLコンパイラでは、メモリ・モデルがスモール・モデル、ミディアム・モデルどちらでも使用可能です。

アセンブリコード使用のオプションを指定した場合は、アセンブラオプションにもチャネルコンパイラ・オプション”__LIN_CHn_Pn__”とアセンブラ版メモリコピー指定”__LIN_MEMCOPY_ASM__”を指定する必要があります。

※3　__LIN_HW_AUTO_BR__を使用する場合、ビット・サンプリング数は4または8に設定する必要があります。
__LIN_HW_AUTO_BR__を使用しない場合、ビット・サンプリング数は16に設定する必要があります。
詳細は「5. 1. 3 confslin_opt.h編集（スレーブ用）」「7. 9. 1 スレーブ・ドライバ・コンフィグレーション」を参照してください。

※4　LINコンフィギュレータによるボー・レート設定に合わせて設定する必要があります。
詳細は「6. 2. 5 ボー・レート設定」を参照してください。

5. 1. 2　confmlin_opt.h 編集（マスター用）

RL78/F23,F24 マスター・ドライバの代表的なコンフィグレーション項目を以下に示します。これらは LIN コンフィギュレータおよびスマート・コンフィグレータで設定できないため、初期設定から変更したい場合は confmlin_opt.h（格納フォルダ libsrc/conf）を直接編集する必要があります。

設定の詳細およびその他のコンフィグレーション項目は「7. 9. 2 マスター・ドライバ・コンフィグレーション」を参照してください。

・CONFMLIN_OPT_u1gLINMCK_CFG　LIN通信クロック源種別（f_{CLK}, f_{MX}）（初期設定 f_{CLK}）
・CONFMLIN_OPT_u1gBDT_CFG　送信するブレーク・デリミタの幅（初期設定 2Tbit）
・CONFMLIN_OPT_u1gIBHS_CFG　送信するヘッダ内のシンク・フィールドとIDフィールド間の幅と、レスポンス送信時のレスポンス・スペースの幅（初期設定 0Tbit）
・CONFMLIN_OPT_u1gIBS_CFG　送信インター・バイト・スペース幅（初期設定 0Tbit）

スマート・コンフィグレータを使用する場合は、以下の項目が出力ヘッダ・ファイル r_rlin3_config.h 内の定義に従って設定されます。使用しない場合は初期設定の値になるため、必要に応じて confmlin_opt.h を直接編集してください。

・CONFMLIN_OPT_u1gBUSWKUP_CFG　バスウェイクアップ方法（初期設定 LINバスの立ち下がりエッジによるウェイクアップ）
・CONFMLIN_OPT_u1gINTXXXPR_CFG
（INTXXX: INTLINTRM, INTLINRVC, INTLINSTA, INTPの4種）　各割り込みの優先度（初期設定 レベル3 (最低)）

5. 1. 3　confslin_opt.h 編集（スレーブ用）

RL78/F23,F24 スレーブ・ドライバの代表的なコンフィグレーション項目を以下に示します。これらは LIN コンフィギュレータおよびスマート・コンフィグレータで設定できないため、初期設定から変更したい場合は confslin_opt.h（格納フォルダ libsrc/conf）を直接編集する必要があります。

コンパイル・オプション__LIN_HW_AUTO_BR__を外して固定ボー・レートで使用する場合は、ビット・サンプリング数 CONFSLIN_OPT_u1gNSPB_NORM_CFG を 16 に変更してください。

設定の詳細およびその他のコンフィグレーション項目は「7. 9. 1 スレーブ・ドライバ・コンフィグレーション」を参照してください。

・CONFSLIN_OPT_u1gLINMCK_CFG　LIN通信クロック源種別（f_{CLK}, f_{MX}）（初期設定 f_{CLK}）
・CONFSLIN_OPT_u1gLPRS_NORM_CFG　LINマクロに供給するプリスケーラ・クロックの分周値（固定ボー・レート時のみ有効）（初期設定 1/1）
・CONFSLIN_OPT_u1gNSPB_NORM_CFG　ビット・サンプリング数（初期設定 4）
・CONFSLIN_OPT_u1gRS_CFG　送信時のレスポンス・スペース幅（初期設定 1Tbit）
・CONFSLIN_OPT_u1gIBS_CFG　送信時のインター・バイト・スペース幅（初期設定 0Tbit）

スマート・コンフィグレータを使用する場合は、以下の項目が出力ヘッダ・ファイル r_rlin3_config.h 内の定義に従って設定されます。使用しない場合は初期設定の値になるため、必要に応じて confslin_opt.h を直接編集してください。

・CONFSLIN_OPT_u1gBUSWKUP_CFG　バスウェイクアップ方法（初期設定 LINバスの立ち下がりエッジによるウェイクアップ）
・CONFSLIN_OPT_u1gINTXXXPR_CFG（INTXXX: INTLINTRM, INTLINRVC, INTLINSTA, INTPの4種）　各割り込みの優先度（初期設定 レベル0 (最高)）
・CONFSLIN_OPT_u1gTMUNIT_CFG他　使用するインターバル・タイマの設定4項目（初期設定 TAU0, チャネル0, CK01使用, タイマ割り込みの優先度レベル0 (最高)）

5. 1. 4　IAR I/O ヘッダ・ファイル指定

RL78/F23、F24のLIN 2.1ソフトウェア・ドライバは、CC-RLコンパイラの他にIARコンパイラにも対応しています

LINコンフィギュレータから出力されるIARコンパイラ向けプロジェクト・ファイルをビルドする際は、IAR社が提供するSFRレジスタ、拡張SFRレジスタ、割り込みベクタなどのアドレス定義や固有関数の宣言がされたヘッダ・ファイルのインクルードを解決する必要があります。

R7F124FPJの場合は以下のファイルになります。

・ior7f124fpj.h

・ior7f124fpj_ext.h

・intrinsics.h

これらのファイルはIAR環境インストール・フォルダに含まれます。また、スマート・コンフィグレータのr_bsp出力ファイルを使用することも可能です。詳細はr_bspのアプリケーションノートを参照してください。

以下のどちらかを行い、device.h（格納フォルダ libsrc/dev）のインクルードを解決してください。

a)　device.hを編集し、#includeで指定されたヘッダ・ファイル名をデバイスに合わせる。
　このとき、パス"../../liblin2/"を含めて記述する。
　指定したファイルをliblin2フォルダにコピーする。

b)　device.hを編集し、#includeで指定されたヘッダ・ファイル名をデバイスに合わせる。
　パス"../../liblin2/"は削除する。
　プロジェクト・ファイルの「標準のインクルードディレクトリを無視」のチェックを外す。

図5-2　device.h 内インクルード記載方法

```
（省略）

/* ---------------------------------------------------------------------- */
/* デバイスファイル                                                       */
/* ---------------------------------------------------------------------- */
#ifdef __CCRL__
#include "../../liblin2/iodefine.h"
#endif /* __CCRL__ */

#ifdef __IAR_SYSTEMS_ICC__
#include "../../liblin2/ior7f124fpj.h"
#include "../../liblin2/ior7f124fpj_ext.h"
#include "../../liblin2/intrinsics.h"
#endif /* __IAR_SYSTEMS_ICC__ */

（省略）
```

左記は a)の場合であり、
b)の場合は、以下のように記載する。

```
#include "ior7f124fpj.h"
#include "ior7f124fpj_ext.h"
#include "intrinsics.h"
```

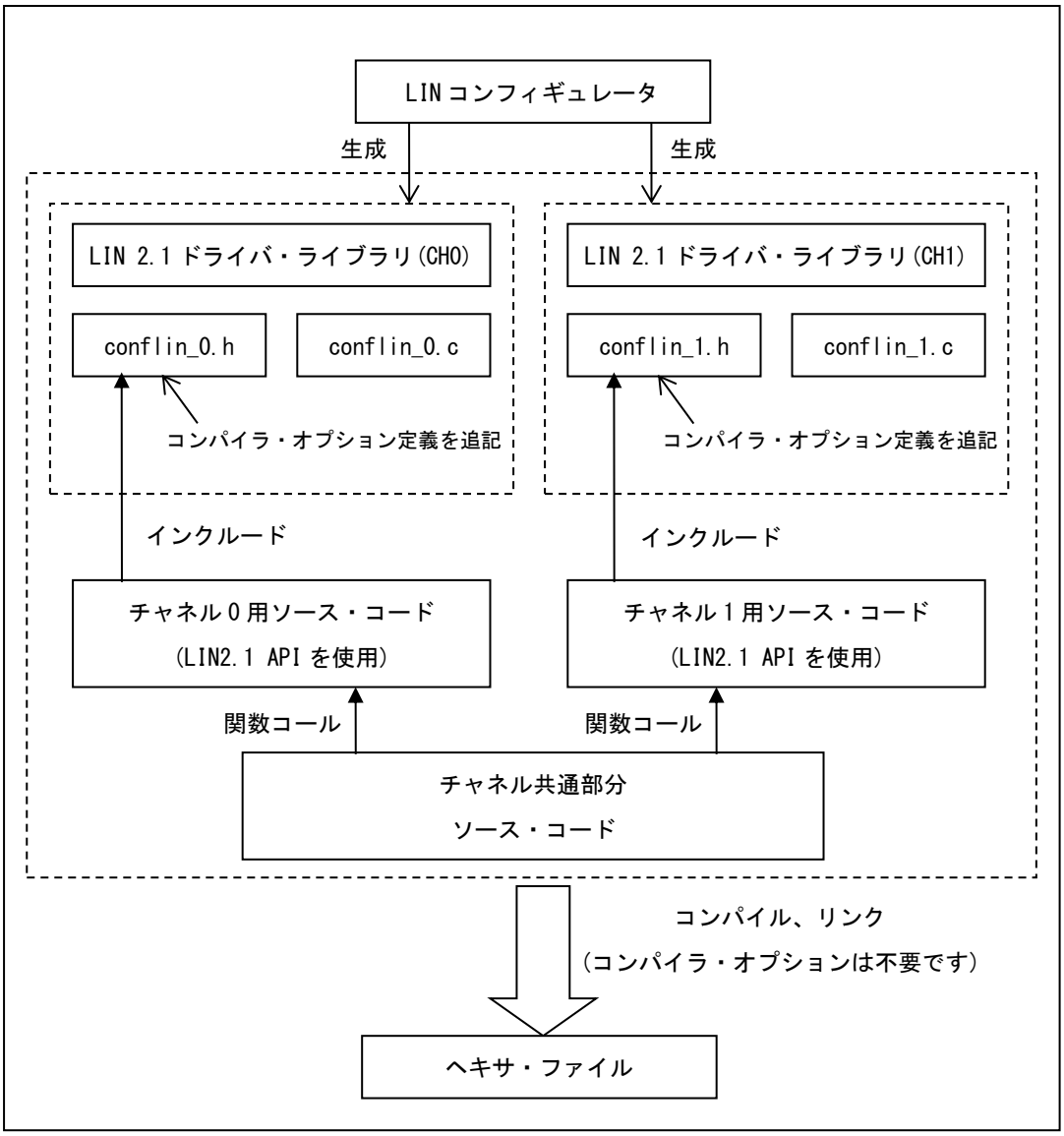
5.2 LIN アプリケーション・プログラム作成

LIN2.1 ソフトウェア・ドライバによって提供される LIN2.1 API を使用して LIN アプリケーション・プログラムを作成します。

以下に、チャネル 0 とチャネル 1 を使用する場合の LIN アプリケーションの構築方法を示します。

1 つのソース・コード(*.c ファイル)が複数の conflin_x.h をインクルードすることはできないため、複数チャネルを使用する場合はチャネル専用ソース・コードを分離する必要があります。使用チャネルが 1 チャネルの場合は、チャネル専用ソース・コードは必須ではありません。

図5-3 LINアプリケーションの構築方法



5. 2. 1　開発環境プロジェクト・ファイル作成

LIN アプリケーション・プログラムの開発環境プロジェクト・ファイルを作成する際、LIN ドライバを使用するために、以下のファイルをビルド対象として追加してください。

・LIN コンフィギュレータで出力した情報ファイル conflin_x.c , conflin_x.h
・LIN ドライバ・ソースをビルドして作成したライブラリ・ファイル（「5. 1　LIN ドライバ・ライブラリ生成」参照）
　liblin21m_CCRL_0.lib、liblin21m_IAR_0.a など
・IAR の場合、リンカ設定ファイル（拡張子 icf）

スマート・コンフィグレータと連携する場合、conflin_x.c , conflin_x.h が空ファイルで生成されるため、LIN コンフィギュレータの出力ファイルで上書きしてください。

LIN ドライバは固有セクションを使用しているため、IAR 環境ではプロジェクト独自のリンカ設定ファイルを追加し、セクションを記述する必要があります。CS+環境ではプロジェクトのプロパティを編集します。指定セクションの詳細は、「5. 2. 8 使用セクションの設定」を参照してください。
なお、コンパイラ・オプション“__LIN_MEMCOPY_ASM__”を使用する場合は、far 領域および huge 領域への配置を行わないでください。

5. 2. 2　チャネル専用ソース・コード作成

LIN2.1 ソフトウェア・ドライバの関数インタフェースは、マスターとスレーブで関数名が共通化されており、チャネルについても関数名が共通化されています。一方、ドライバ・ライブラリは、マスター／スレーブとチャネル番号で個別に関数が実装されています。
このため、共通化された関数インタフェースとドライバ・ライブラリの関数名とを結びつけるために、ヘッダ・ファイル conflin_x.h をインクルードする必要があります。
例　関数名 l_u16_rd
　conflin_0.h（マスター用）をインクルード → ApMLin_u2gRead16bitsSig_0 に置き換え
　conflin_1.h（スレーブ用）をインクルード → ApSLin_u2gRead16bitsSig_1 に置き換え

1 つのソース・コード(*.c ファイル)が複数の conflin_x.h をインクルードすることはできません。
したがって、複数のチャネルを使用する LIN アプリケーション・プログラムを作成する場合、チャネル毎にソース・コードを用意し、それぞれが conflin_x.h をインクルードする必要があります。
チャネル毎のソース・コード内でそのチャネル用の LIN2.1API を使用してください。

5. 2. 3　周辺ハードウェア処理実装

LIN ドライバは 1. 4. 3 に示す周辺ハードウェアの操作を行います。その他の周辺ハードウェアの設定は行わないため、LIN アプリケーションを作成する際に、ユーザ・コードによる周辺ハードウェア処理が必要です。

主な周辺ハードウェア処理は以下の通りです。

- クロック発生回路の設定

  LIN2.1ソフトウェア・ドライバ内部では、クロック発生機能に関する設定は行っておりません。アプリケーションにてこれらの設定を行ってください。

  スマート・コンフィグレータの出力ソースとリンクする場合は、ユーザ・コード不要です。

- LINトランシーバの設定

  LIN2.1 ソフトウェア・ドライバでは、LIN トランシーバの操作を行いません。アプリケーションにてこれらの設定を行ってください。

  スマート・コンフィグレータの出力ソースとリンクする場合は、ポート機能を使用することにより出力端子の初期設定が可能です。端子のレベル切り替えにはユーザ・コードが必要です。

- スタンバイ機能

  LIN2.1ソフトウェア・ドライバ内部では、スタンバイ･モード移行時の制御は行っておりません（HALT、STOP、SNOOZEモード等）。必要な場合は、アプリケーション部にて設定を行ってください。

  RL78/F23,F24 では、ドライバのウェイクアップ方法が「立ち下がりエッジ検出」の場合のみドライバのスリープ中にスタンバイ・モードへの移行が可能です(ウェイクアップ方法はドライバ・コンフィグレーションで選択します)。マスターでは l_ifc_goto_sleep()発行後の l_ifc_read_status() の戻り値が”0x3C02”（bit3-7 は不定値）であることを確認してからスタンバイ・モードへ移行してください。スレーブでは l_sys_call_wake_up()がコールされた後でスタンバイ・モードへ移行してください。

LIN アプリケーションを作成する際の周辺ハードウェアに関する注意事項は以下の通りです。

- 周辺ハードウェア使用上の注意

  LIN2.1ソフトウェア・ドライバで使用している資源はユーザ･アプリケーションでは使用することができません。LIN2.1ソフトウェア・ドライバの使用資源は、1. 4. 3を参照してください。

- LIN 使用ポートの出力ラッチについて

  RL78 LIN2.1ソフトウェア・ドライバでは使用チャネルのTXD端子が割り当たっているポートの出力ラッチを操作するため、他のポートレジスタビットが入力設定の場合、その出力ラッチを入力端子レベルの値に書き換えることに注意してください。

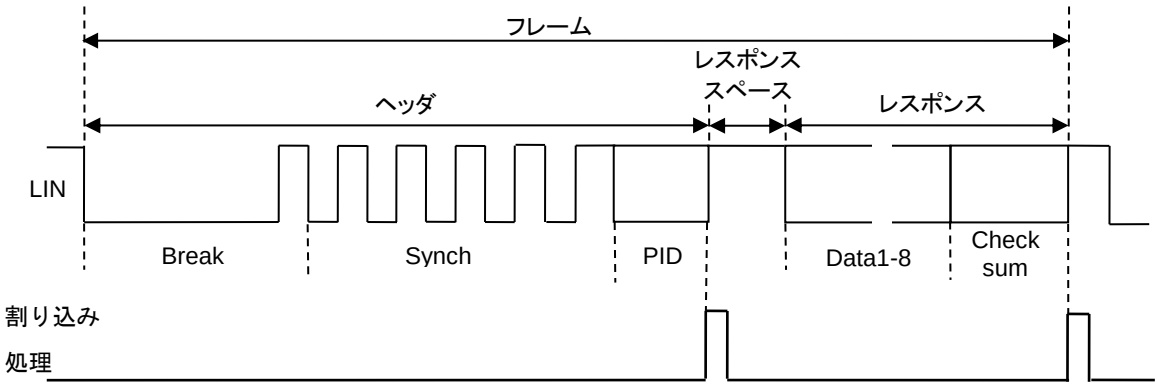
- 割り込み優先順位の設定

  LIN2.1ソフトウェア・ドライバの割り込み優先度はドライバ・コンフィグレーション、またはスマート・コンフィグレータで設定します。ユーザ･アプリケーションで割り込みを利用する場合は、LIN2.1ソフトウェア・ドライバが使用する割り込みとの競合に注意する必要があります。

  スレーブ･ドライバで使用する割り込みを高優先度（ユーザ･アプリケーションで使用する割り込みは低優先度）に設定し、多重割り込みを許可することを推奨いたします。

  マスター・ドライバでは、ドライバのスリープ、ウェイクアップ以外では割り込みを使用しません。このため、高優先度の設定は不要です。

図5-4　RL78/F23,F24のスレーブ割り込み処理タイミングイメージ



・PID受信後の割り込み処理はData1を受信完了するまでに処理完了される必要があります。レスポンス送信の場合、レスポンス・スペースは伸びますが、フレーム長がLIN規定内に収まるように考慮してください。

・Check Sum送受信後の割り込み処理は、次フレームのBreakを受信完了するまでに処理を完了してください。

5. 2. 4　コンパイラ・オプション編集（conflin_x.h）

LIN コンフィギュレータによって生成された conf/conflin_x.h (“x”はチャネル番号)の先頭部分にコンパイラ・オプションを記述します。これはチャネル毎に行う必要があります。

LIN アプリケーションに必要なコンパイラ・オプションは以下のとおりです。このうち、対象マイコン、チャネルは LIN コンフィギュレータが自動指定を行います。__LIN_SIGNAL_32__を LIN ドライバ・ライブラリ生成時に追加した場合は conflin_x.h を直接編集する必要があります。(「5. 1. 1 ライブラリ用コンパイラ･オプション」参照)

表 5-2　コンパイラ・オプション（conflin_x.h）一覧

コンパイラ・オプション	マクロ名	RL78/F23,F24 マスター	RL78/F23,F24 スレーブ
対象マイコン	__LIN_RL78_F23_F24__	LINコンフィギュレータが自動指定（削除不可）	
チャネル	__LIN_CH0__または __LIN_CH1__	LINコンフィギュレータが自動指定（削除不可） conflin_0.h: __LIN_CH0__自動指定 conflin_1.h: __LIN_CH1__自動指定	
フレーム内信号数32	__LIN_SIGNAL_32__	LINドライバ・ライブラリ生成時に 指定している場合は、追加が必要	

conflin_x.h 内のマクロ記載位置と記載内容の例は以下になります(コメント”記載例”の太字部分)。

図5-5　conflin_x.h 内マクロ記載方法

```
（省略）

#ifndef H_CONFLIN
#define H_CONFLIN

/* ********************************************************************** */
/* 【アプリケーション用マクロ定義を記載してください】(ここから)          */
/*                                                                        */
/* [例]                                                                   */
/*   #define __LIN_CH0__                                                  */
/*                                                                        */
/* [補足]                                                                 */
/*   ・記載内容はデバイス毎に異なります。詳細は readme.txt の「コンパイ */
/*     ルオプション」の項目を参照してください。                           */
/* ********************************************************************** */

#define __LIN_RL78_F23_F24__
#define __LIN_CH0__
#define __LIN_SIGNAL_32__       /* 記載例 */

/* ********************************************************************** */
/* 【アプリケーション用マクロ定義を記載してください】(ここまで)          */
/* ********************************************************************** */

（省略）
```

5. 2. 5　公開定数編集（conflin_x.c）

LIN コンフィギュレータによって生成された conf/conflin_x.c (“x”はチャネル番号)には const 変数による公開定数が含まれます。通常、confiln_x.c は編集不要ですが、以下の場合には編集が必要です。

a) LIN 通信クロックに、LIN コンフィギュレータで指定可能な周波数（8, 10, 12, 16, 20, 24, 32, 40 [MHz]）以外を使用する場合

LIN コンフィギュレータで指定できない LIN 通信クロック周波数を使用する場合は、下記のように CONFLIN_u2sPERICLOCK の定義値に周波数（分解能 10kHz）を指定します。

例　LIN 通信クロックに 4MHz を使用

【修正前】　LIN 通信クロック 40 MHz = 10 kHz x 4000

```
#define CONFLIN_u2sPERICLOCK            (4000)
```

【修正後】　LIN 通信クロック 4 MHz = 10 kHz x 400

```
#define CONFLIN_u2sPERICLOCK            (400)
```

b) 以下の条件を満たす場合

・スレーブ・ドライバを使用　かつ

・LIN の供給クロックに f_{CLK} ではなく f_{MX} を使用　かつ

・f_{CLK} と f_{MX} で周波数が異なる

LIN スレーブ・ドライバの初期設定は、LIN とインターバル・タイマ(TAU)の供給クロックがともに f_{CLK} で同一の周波数が入力されることを前提としています。

このため、LIN 供給クロックに f_{MX} を使用することにより、TAU と異なる周波数が供給される場合は、LIN コンフィギュレータでコード生成後、下記のように TAU 供給クロック周波数（分解能 10kHz）を示す定数 ConfSLin_u2gTMPERICLOCK を修正する必要があります。

例　LIN 通信クロック 8MHz (f_{MX})、TAU 供給クロック 40MHz (f_{CLK}) を使用

【修正前】　LIN 通信クロック、TAU 供給クロックともに 8 MHz = 10 kHz x 800

```
#define CONFLIN_u2sPERICLOCK            (800)
const u2 ConfSLin_u2gPERICLOCK          = (u2)CONFLIN_u2sPERICLOCK;
const u2 ConfSLin_u2gTMPERICLOCK        = (u2)CONFLIN_u2sPERICLOCK;
```

【修正後】　LIN 通信クロック 8MHz、TAU 供給クロック 40 MHz = 10 kHz x 4000

```
#define CONFLIN_u2sPERICLOCK            (800)
const u2 ConfSLin_u2gPERICLOCK          = (u2)CONFLIN_u2sPERICLOCK;
const u2 ConfSLin_u2gTMPERICLOCK        = (u2)4000;
```

5.2.6 スケジューラの実装（マスターのみ）

LIN マスターのアプリケーションを作成する場合、マスターのユーザ･アプリケーション内でタイマを用い、スケジューラを実装する必要があります。

スケジューラの詳細は、「7.7 スケジューリング機能 【マスター】」を参照してください。

以下にスマート・コンフィグレータのタイマ機能を用いたスケジューラ設定のプログラミング例を示します。

・コンポーネント：インターバル・タイマ

・コンフィグレーション名：Config_TAU0_1

・動作モード：16 ビット・カウンタ・モード

・リソース：TAU0_1

・出力ファイル：Config_TAU0_1.c, Config_TAU0_1.h, Config_TAU0_1_user.c

[Config_TAU0_1_user.c] 抜粋

・Include 対象に conflin_0.h を追加

```
/*****************************************************************************************************************
Includes
*****************************************************************************************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_userdefine.h"
#include "Config_TAU0_1.h"
/* Start user code for include. Do not edit comment generated here */
#include "conflin_0.h"
/* End user code. Do not edit comment generated here */
```

・割り込みルーチンに l_sch_tick を追加

[CC-RL]

```
/*****************************************************************************************************************
* Function Name: r_Config_TAU0_1_interrupt
* Description   : This function is INTTM01 interrupt service routine.
* Arguments     : None
* Return Value : None
*****************************************************************************************************************/
static void __near r_Config_TAU0_1_interrupt(void)
{
    /* Start user code for r_Config_TAU0_1_interrupt. Do not edit comment generated here */
    (void)l_sch_tick(LIN_CHANNEL0);
    /* End user code. Do not edit comment generated here */
}
```

[IAR]

```
#pragma vector = INTTM01_vect
__interrupt static void r_Config_TAU0_1_interrupt(void)
{
    /* Start user code for r_Config_TAU0_1_interrupt. Do not edit comment generated here */
    (void)l_sch_tick(LIN_CHANNEL0);
    /* End user code. Do not edit comment generated here */
}
```

[main.c]

・Include 対象にタイマ機能ヘッダ Config_TAU0_1.h を追加

・Include 対象に conflin_0.h を追加

・LIN 初期化後、タイマスタート関数 R_Config_TAU0_1_Start 呼び出し

・関数 l_sch_set でスケジュール・テーブルをセットする

```
#include "platform.h"
#include "Config_TAU0_1.h"

/* defined in configuration file.
            - Channel name      : LIN_CHANNEL0
            - Schedule name     : SCH_MAINAPL
*/
#include "conflin_0.h"

/* main function */
void main( void )
{
    /* LIN initialize or more */

    BSP_ENABLE_INTERRUPT();

    R_Config_TAU0_1_Start();                    /* Start timer                    */

    l_sch_set(LIN_CHANNEL0, SCH_MAINAPL, 0U);   /* Set schedule of main app.      */
    ApLst_vosMainApl();                         /* Start main application         */
}
```

5. 2. 7　ユーザ定義コール・アウトの実装

LIN2.1ソフトウェア・ドライバは、割り込み禁止と復帰、ウェイクアップ通知などを行うため、コール・アウト関数の呼び出しを行います。LINドライバ・ライブラリとリンクするためには各コール・アウト関数をユーザ・コードで実装する必要があります。

スマート・コンフィグレータを使用してコード出力した場合は、必要なコール・アウト関数として以下のソース・ファイルが出力されます。

マスター・ドライバ用ソース・コード r_rlin3_m_callout.c

スレーブ・ドライバ用ソース・コード r_rlin3_s_callout.c

詳細は「8. 3. 8 [スレーブ] ユーザ定義コール・アウト（4種類）」「9. 3. 10 [マスター] ユーザ定義コール・アウト（4種類）」を参照してください。

5. 2. 8　使用セクションの設定

LIN2.1ソフトウェア・ドライバは、使用するコード領域、データ領域を固有セクションに配置します。そのため、リンクするためにはセクションの定義が必要です。

CS+の場合はプロジェクト・ファイルに定義し、IARの場合はリンカ設定ファイル（拡張子icf）に対象セクションを追記してリンクします。編集方法の詳細は、使用する統合開発環境のユーザーズ・マニュアルを参照してください。

（1）CC-RL 向け使用セクション

CS+プロジェクト・ファイルで、以下のセクションを定義してください。

マスター・ドライバ

・ROM 領域：LMCODE_n, LMCODE_f, LMCNST_n, LMCNST_f

・RAM 領域：LMDATA_n, LMDATA_f

スレーブ・ドライバ

・ROM 領域：LSCODE_n, LSCODE_f, LSCNST_n, LSCNST_f

・RAM 領域：LSDATA_n, LSDATA_f

表 5-3　LIN2.1 ソフトウェア・ドライバ使用セクション（CC-RL）

デフォルト・セクション名	内容	LIN ドライバ使用セクション名		備考
		マスター	スレーブ	
.callt0	callt 関数呼び出しのテーブル用セクション	（未使用）	（未使用）	
.text	コード部用セクション（near 領域配置）	LMCODE_n	LSCODE_n	関数、割り込み関数
.textf	コード部用セクション（far 領域配置）	LMCODE_f	LSCODE_f	
.textf_unit64kp	コード部用セクション（セクションを先頭が偶数番地になるように，64KB-1 境界にまたがらないように配置）	（未使用）	（未使用）	
.const	ROM データ（near 領域配置）（ミラー領域内）	LMCNST_n	LSCNST_n	定数データ（グローバル、static 含む）
.constf	ROM データ（far 領域配置）	LMCNST_f	LSCNST_f	
.data	near 初期化データ用セクション（初期値あり）	（未使用）	（未使用）	
.dataf	far 初期化データ用セクション（初期値あり）	（未使用）	（未使用）	
.sdata	初期化データ用セクション（初期値あり，saddr 配置変数）	（未使用）	（未使用）	
.bss	データ領域用セクション（初期値なし，near 領域配置）	LMDATA_n	LSDATA_n	初期値なし変数データ（グローバル、static 含む）
.bssf	データ領域用セクション（初期値なし，far 領域配置）	LMDATA_f	LSDATA_f	
.sbss	データ領域用セクション（初期値なし，saddr 配置変数）	（未使用）	（未使用）	
.option_byte	ユーザ・オプション・バイト，およびオンチップ・デバッグ指定専用セクション	（未使用）	（未使用）	
.security_id	セキュリティ ID 指定専用セクション	（未使用）	（未使用）	
.vect< ベクタテーブル・アドレス >	割り込みベクタ・テーブル	–	–	

※注意：

コンパイラ・オプション“__LIN_MEMCOPY_ASM__”を使用する場合は、far 領域を使用する配置は行わないでください。

（2）IAR 向け使用セクション

IAR のリンカ設定ファイル（拡張子 icf）で、以下のセクションを定義してください。

マスター・ドライバ

・ROM 領域：LMCODE, LMCNST

・RAM 領域：LMDATA

スレーブ・ドライバ

・ROM 領域：LSCODE, LSCNST

・RAM 領域：LSDATA, LSDATA

表 5-4　LIN2.1 ソフトウェア・ドライバ使用セクション（IAR）

配置対象	デフォルト・セクション名	LIN ドライバ使用セクション名		補足
		マスター	スレーブ	
関数、割り込み関数、_callt 関数	.text	LMCODE	LSCODE	near 領域用
	.textf	LMCODE	LSCODE	far 領域用
定数データ	.const	LMCNST	LSCNST	near 領域用
	.constf	LMCNST	LSCNST	far 領域用
	.consth	LMCNST	LSCNST	huge 領域用
割り込みベクタ・テーブル	.intvec	–	–	–
初期化される静的/グローバル変数	.data	（未使用）	（未使用）	near 領域用
	.dataf	（未使用）	（未使用）	far 領域用
	.hdata	（未使用）	（未使用）	huge 領域用
0 に初期化される静的/グローバル変数	.bss	LMDATA	LSDATA	near 領域用
	.bssf	LMDATA	LSDATA	far 領域用
	.hbss	LMDATA	LSDATA	huge 領域用
.data セクションの初期値	.data_init	（未使用）	（未使用）	near 領域用
	.dataf_init	（未使用）	（未使用）	far 領域用
	.hdata_init	（未使用）	（未使用）	huge 領域用
_callt の使用によって生成される呼出しテーブルベクタ	.callt0	（未使用）	（未使用）	–
OCD オプションバイト	.option_byte	（未使用）	（未使用）	–
セキュリティ ID	.security_id	（未使用）	（未使用）	–

※注意：

コンパイラ・オプション“__LIN_MEMCOPY_ASM__”を使用する場合は far 領域または huge 領域を使用する配置は行わないでください。

第6章 LIN コンフィギュレータ

6. 1　概要

LIN コンフィギュレータは、ユーザが LIN 機能を組み込んだシステム構築において、LIN の初期値を設定するための開発支援ツールです。ユーザが使用するデバイスに即した外部変数の初期設定や、システム中で使用するさまざまなフレーム（メッセージ）の静的生成を行うための機能を提供します。複数の LIN チャネルの同時設定が可能です。

6. 1. 1　特徴

LIN コンフィギュレータは使用するデバイスの選択、周辺ハードウェア・クロック値の入力、ボー・レート値の指定などから、そのデバイスに即した外部変数の初期値を決定することができます。また、使用する無条件フレーム、イベント・トリガ・フレームなどの各種フレーム、スケジュールの設定やノード情報の設定など、LIN 通信に必要な設定を一度に行うことが可能です。

LIN コンフィギュレータでは、プロジェクト・ファイルにより、ユーザが入力した各種情報を管理しています。プロジェクト・ファイルの保存・読み出しにより、途中からの作業再開が可能となります。

6. 1. 2　実行環境

LINコンフィギュレータを動作させる上で必要となる環境は以下の通りです。

ホストマシン　：IBM-PC/AT™ 互換機
OS　：Windows 10
(Microsoft .NET Framework 3.5 が必要です)

6. 1. 3　出力フォルダ

LIN コンフィギュレータは以下のフォルダを生成します。

conf　：コンフィグ･ファイルが格納されています。
liblin2　：統合開発環境のプロジェクト・ファイル等が格納されています。LIN2.1ソフトウェア・ドライバ・ソースからコンパイラ･オプション依存部のライブラリを生成するために使用します。
libsrc　：LIN2.1ソフトウェア・ドライバ・ソース（コンパイラ・オプション依存部）が格納されています。

6. 2　ファイル生成手順

デバイス設定画面での使用デバイス設定が完了すれば、その他は順不同で設定することが可能です。一般的には、6. 2. 1～6. 2. 11 の手順で設定することを推奨します。

ただし、設定値を入力する場合は以下の点に注意してください。

※　全般的な注意事項

・名称（フレーム名、シグナル名、スケジュール・テーブル名）

これらの名称は、重複しないようにしてください。また、名称は C 言語の識別子パターンに則って指定してください。

・数値

基本的に 10 進数で記述してください。ID/Initial ID、Initial Data、NAD、Supplier ID、Function ID、Variant には、C 言語における 16 進数の記述規則にしたがって 16 進数の記述を行うことも可能です。

・XML ファイル

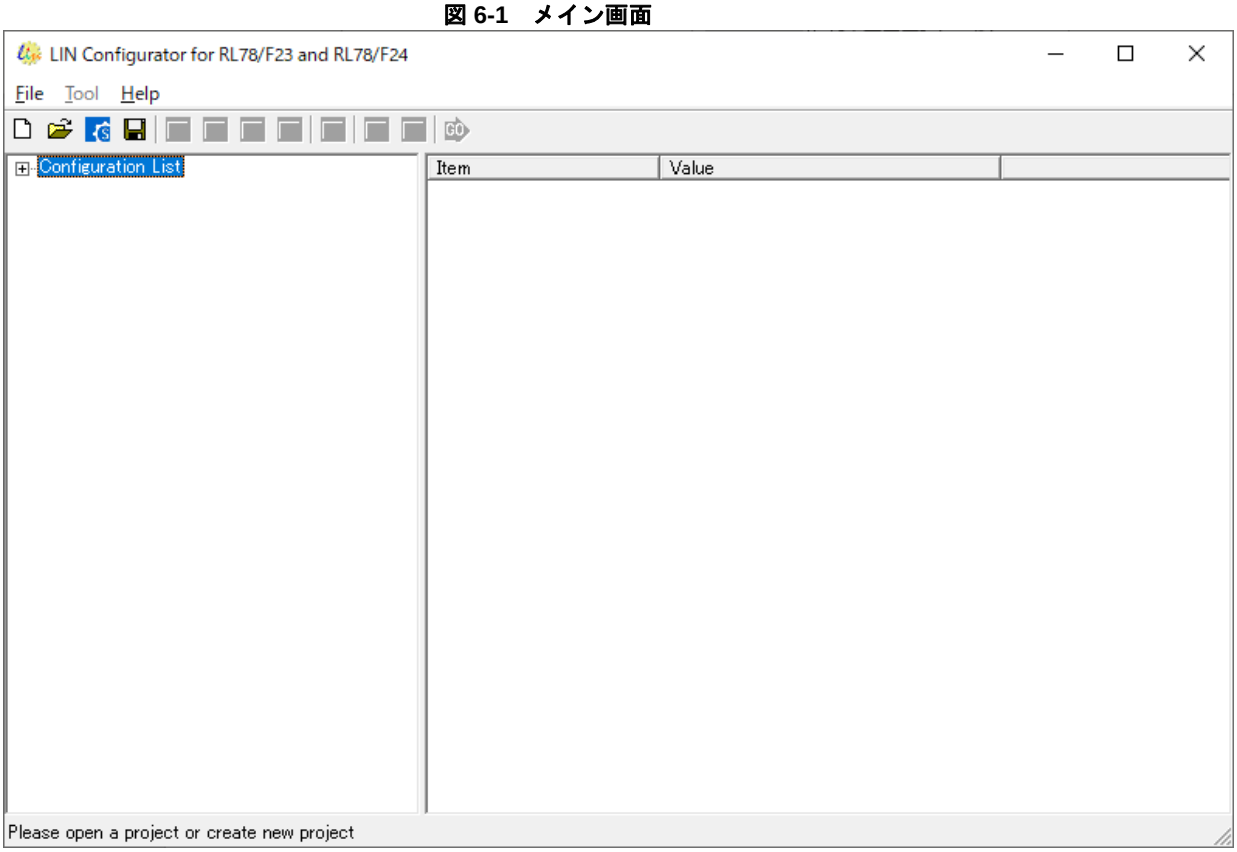
LIN コンフィギュレータに関連する XML ファイルは変更しないでください。正常に動作しなくなる可能性があります。

6. 2. 1　LIN コンフィギュレータの起動

スタートメニューからLINコンフィギュレータを起動すると、以下のようなメイン画面が表示されます。画面の左側には大項目が表示され、右側には項目ごとの設定内容が表示されます。設定されていない項目は表示されません。

(画面はすべて Windows 10 の場合です)

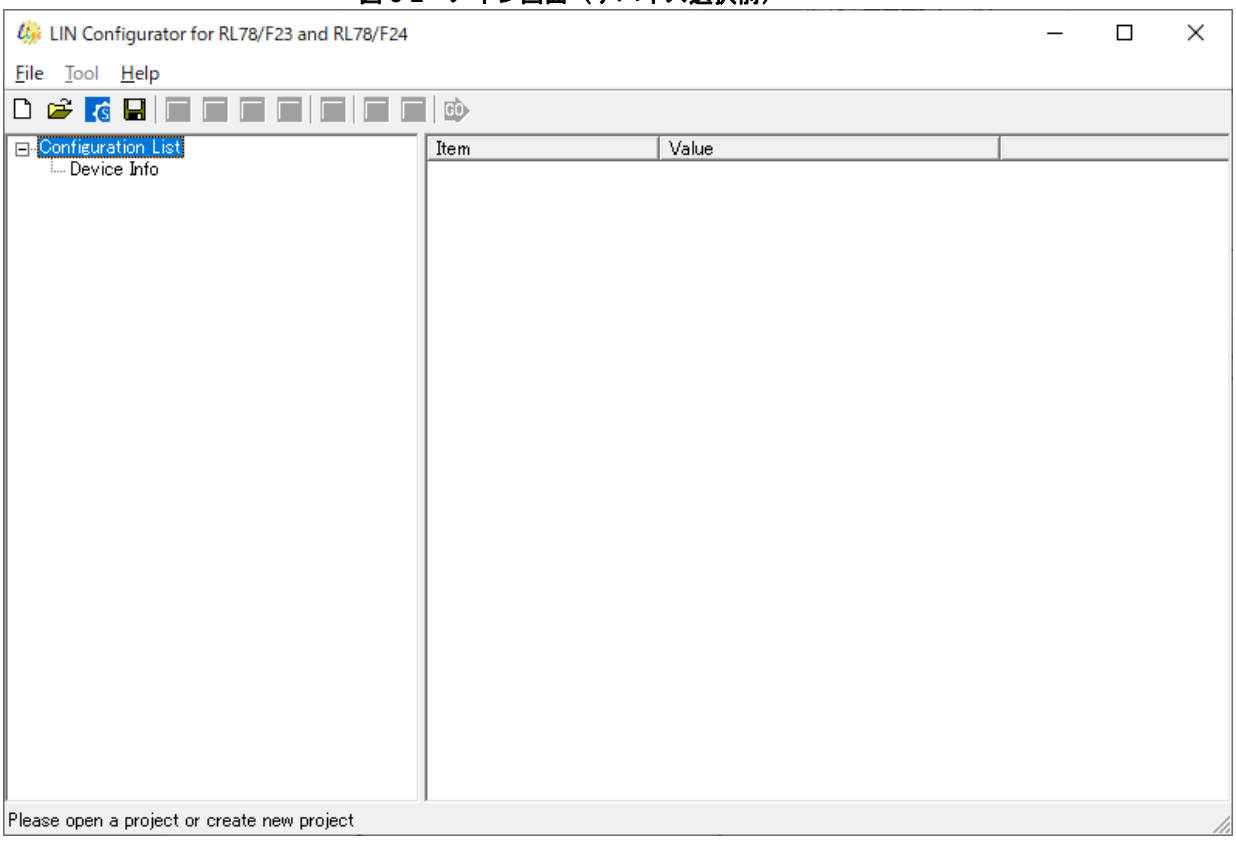
図 6-1　メイン画面



“Configuration List”にはデバイスと各チャネルの設定項目がぶら下がります。

“Configuration List”の左側の”+”をクリックすると下のような画面になります。

図 6-2　メイン画面（デバイス選択前）



上部のメニューから選択できる項目は以下の通りです。

- “File”
  - “New”　‥ それまでの設定をすべて破棄し、新たにコンフィギュレーションを行います。
  - “Open”　‥ 設定が保存された XML ファイルを読み込みます。
  - “SMC Header File Open”　‥ スマート・コンフィグレータが出力した LIN 関連定義ファイル（ヘッダ・ファイル）を読み込みます。（ボタン でも実行可能）
  - “Save”　‥ それまでの設定を前回保存した XML ファイルに上書きします。
  - “Save As...”　‥ それまでの設定を新たな XML ファイルに保存します。
  - “Exit”　‥ LIN Configurator を終了します。
- “Tool”‥“Tool”メニュー配下の項目からの操作は次頁の表を参照してください。
- “Help”
  - “About Configurator...”‥ LIN Configurator の情報が別ウィンドウで表示されます。

また、メイン画面上部のボタンは、選択中のツリー配下のチャネル(デバイス設定後にぶら下がります)のマスター、スレーブの種別によって選択可否が決まります。次頁の表を参照してください。

【注意条項】

“SMC Header File Open”の実行により、それまでに設定された情報はすべてクリアされます。

表 6-1　ツリー項目、ボタン、Tool メニューの操作(マスターチャネル時)

操作			オープンするウィンドウ
ツリー項目(ダブルクリック)	ボタン	Toolメニュー選択	
Device Info	D	Device Setup	**Device Selection (デバイス選択)**
Channel*	C	Channel Setup※1	**Channel Configuration (チャネル設定)**
Baud rate	B	Baud Rate Setup※1	**Setting Baud Rate (ボー・レート設定)**
Unconditional frames	M	Message Setup※1	**Setting Frame (メッセージ管理)**
Unconditional frames **配下のメッセージ**			
Unconditional frames **配下の信号**			
Event triggered frames			
Event triggered frames **配下のメッセージ**			
Sporadic frames			
Sporadic frames **配下のメッセージ**			
Schedules	S	Schedule Setup※1	**Setting Schedule (スケジュール管理)**
Schedules **配下のスケジュール**			

※1：選択するためには、ツリー内でチャネルが選択されている必要があります。ボタンも同様です。

表 6-2　ツリー項目、ボタン、Tool メニューの操作(スレーブチャネル時)

操作			オープンするウィンドウ
ツリー項目(ダブルクリック)	ボタン	Toolメニュー選択	
Device Info	D	Device Setup	**Device Selection (デバイス選択)**
Channel*	C	Channel Setup※2	**Channel Configuration (チャネル設定)**
Baud rate	B	Baud Rate Setup※2	**Setting Baud Rate (ボー・レート設定)**
Unconditional frames	M	Message Setup※2	**Setting Frame (メッセージ管理)**
Unconditional frames **配下のメッセージ**			
Unconditional frames **配下の信号**			
Event triggered frames			
Event triggered frames **配下のメッセージ**			
Node information	N	Node Setup※2	**Setting Node (ノード設定)**
Others	O	Others Setup※2	**Setting Others (その他設定)**

※2：選択するためには、ツリー内でチャネルが選択されている必要があります。ボタンも同様です。

6. 2. 2　新規コンフィギュレーションの開始

新規のコンフィギュレーションを開始するには、以下の2通りがあります。

a)　スマート・コンフィグレータのコンフィグを用いて開始する場合

Fileメニューの“SMC Header File Open”を選択すると、ファイル読み込みのダイアログが表示されます。
スマート・コンフィグレータが出力したLIN関連定義ファイル r_rlin3_config.h を指定してください。
ファイルに設定された情報を初期状態として、新規のコンフィギュレーションが作成されます。
ボタンでも実行可能です。

ファイルの読み出しに成功した場合、スマート・コンフィグレータ上で設定された情報が6. 2. 3 デバイス選択画面と6. 2. 4 チャネル設定画面に自動的に反映されます。
反映される項目は次のとおりです。

[6. 2. 3 デバイス選択画面]

Series Information
Device Name
Channel

[6. 2. 4 チャネル設定画面]

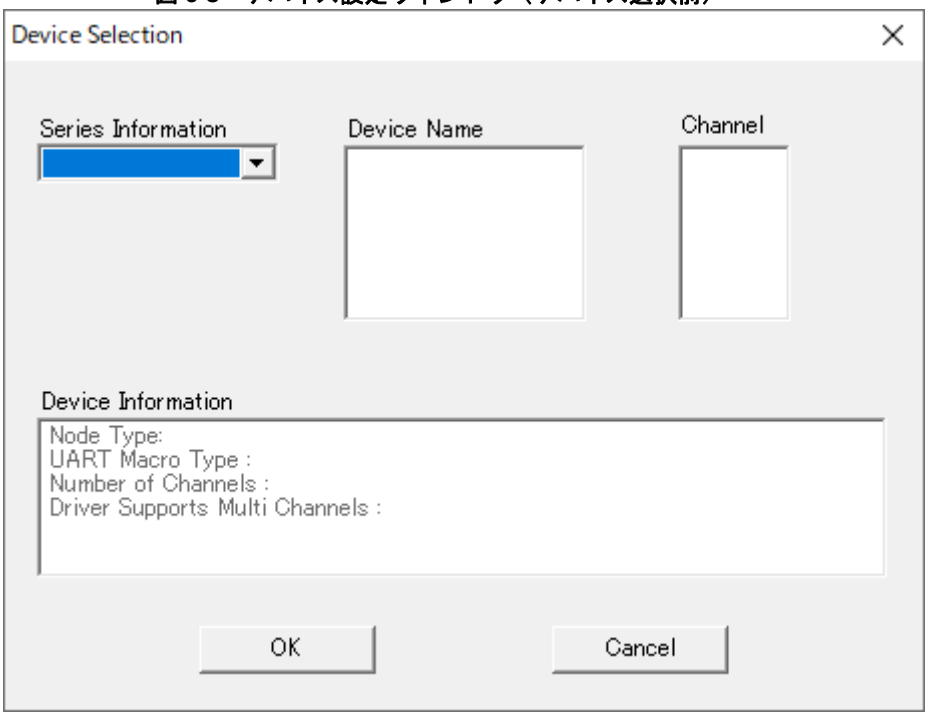
Master/Slave
Peripheral Hardware Clock

b)　LINコンフィギュレータで新規に作成する場合

Fileメニューの”New”を選択すると、6. 2. 3 デバイス選択画面が表示されます。デバイスとチャネルの選択を完了すると、新規のコンフィギュレーションが作成されます。
ボタンでも実行可能です。

6. 2. 3　デバイス選択

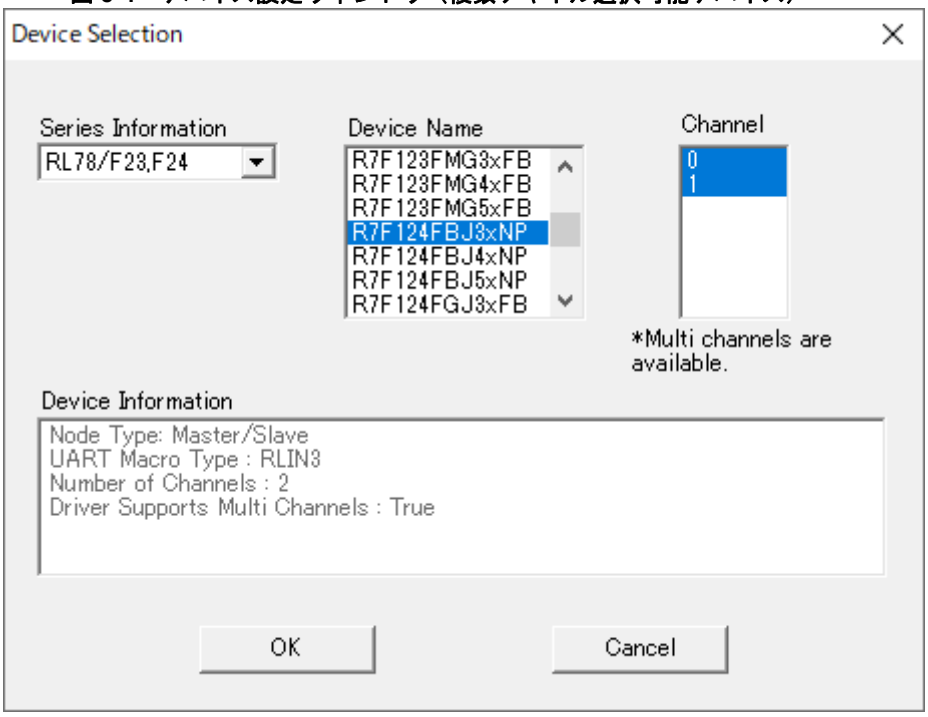
図 6-3　デバイス設定ウィンドウ（デバイス選択前）



このウィンドウでマイコンのシリーズ、デバイスを選択します。

複数チャネル同時選択可能なデバイスの場合、”Channel”リストの下にその旨のメッセージが表示され、”Channel”リスト内のチャネルを複数選択することが可能になります。

図 6-4　デバイス設定ウィンドウ（複数チャネル選択可能デバイス）



【備考】

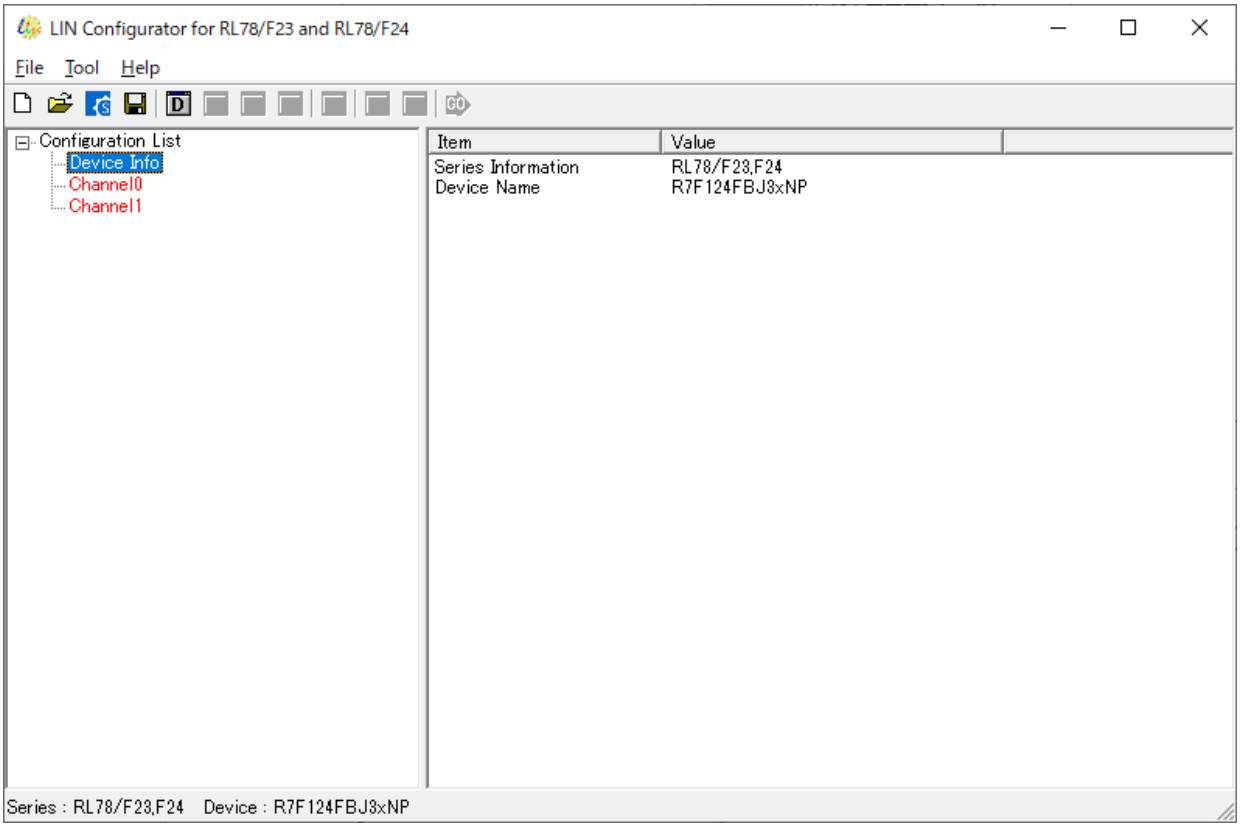
スマート・コンフィグレータから出力された LIN 関連定義ファイルを読み出すことで、スマート・コンフィグレータ上で設定したデバイス、チャネル情報が本画面に反映されます。

チャネルを選択した後、”OK”をクリックしてください。

メイン画面の”Configuration List”の下にすべてのチャネルがぶら下がった状態になります。

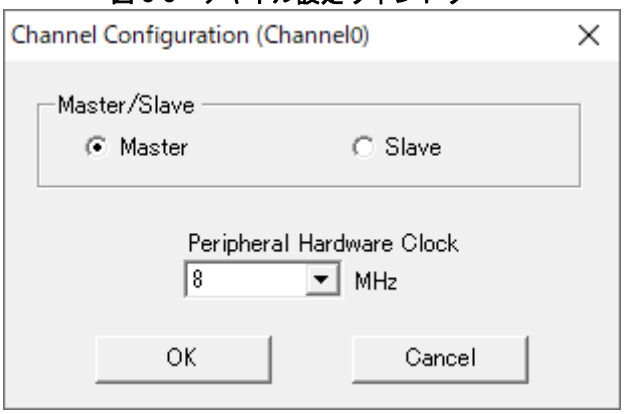
(コンフィギュレーションが完了していないチャネルは赤字で表示されます。)

図 6-5　メイン画面（チャネル設定前）



6. 2. 4　チャネル設定

図 6-6　チャネル設定ウィンドウ



チャネル設定画面では、チャネル毎のマスターとスレーブの切り替え、LIN ドライバが使用するクロック周波数を選択します。設定後、”OK”をクリックしてください。

【備考】

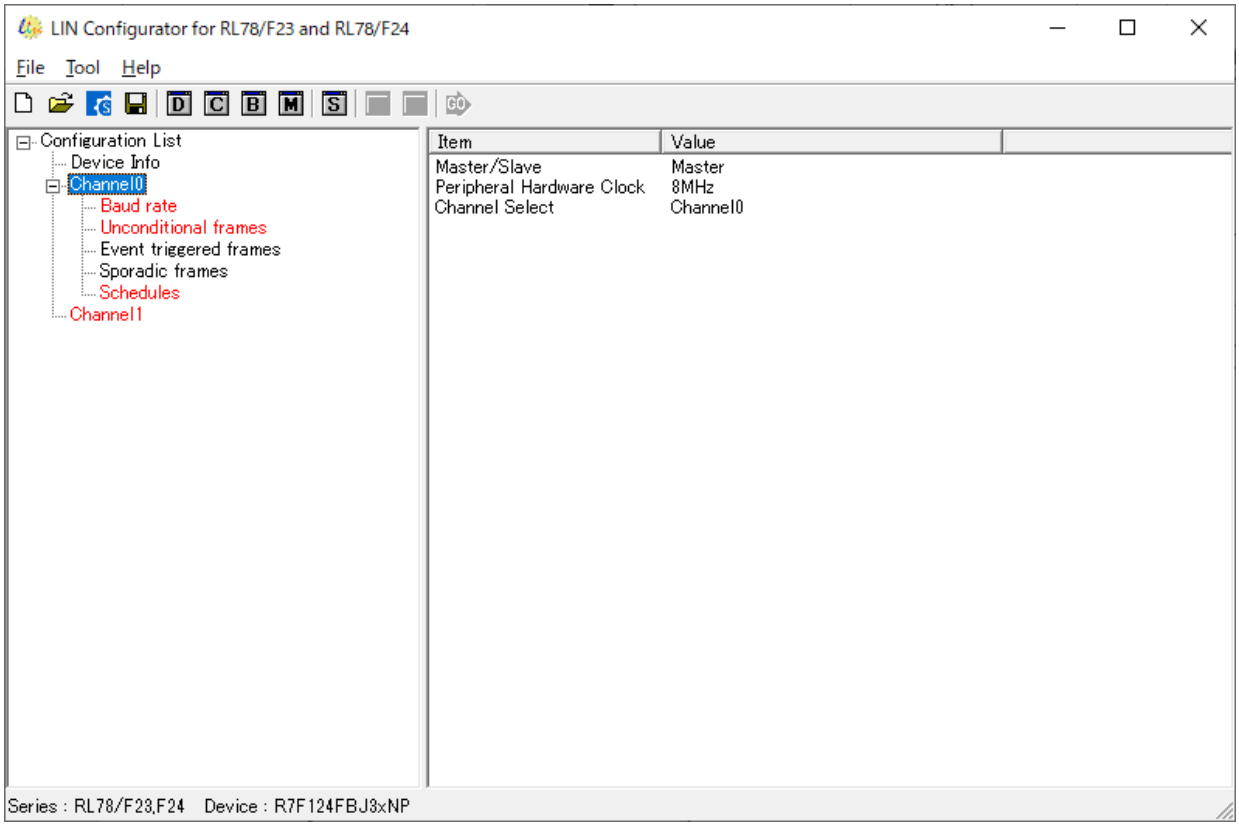
スマート・コンフィグレータから出力された LIN 関連定義ファイルを読み出すことで、スマート・コンフィグレータ上で設定した Master/Slave、入力クロックが本画面に反映されます。

【注意条項】

LIN ドライバ内部ではクロック発生機能に関する設定は行っておりません。アプリケーションにてこれらの設定を行ってください。

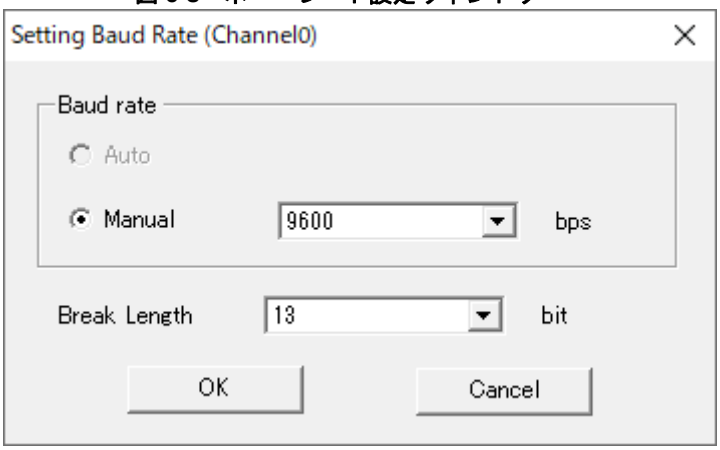
チャネル設定完了後、メイン画面のツリー内のチャネル配下に各項目がぶら下がった状態になります。ドライバのコード出力に必須な項目のうち、未設定の項目は赤字になります。

図 6-7　メイン画面（チャネル項目設定前）



6. 2. 5　ボー・レート設定

図 6-8　ボー・レート設定ウィンドウ



ボー・レートの設定と、送信 Break フィールドのロウ・レベルの長さを設定します。
(送信 Break フィールド長はマスターでのみ設定可能です)

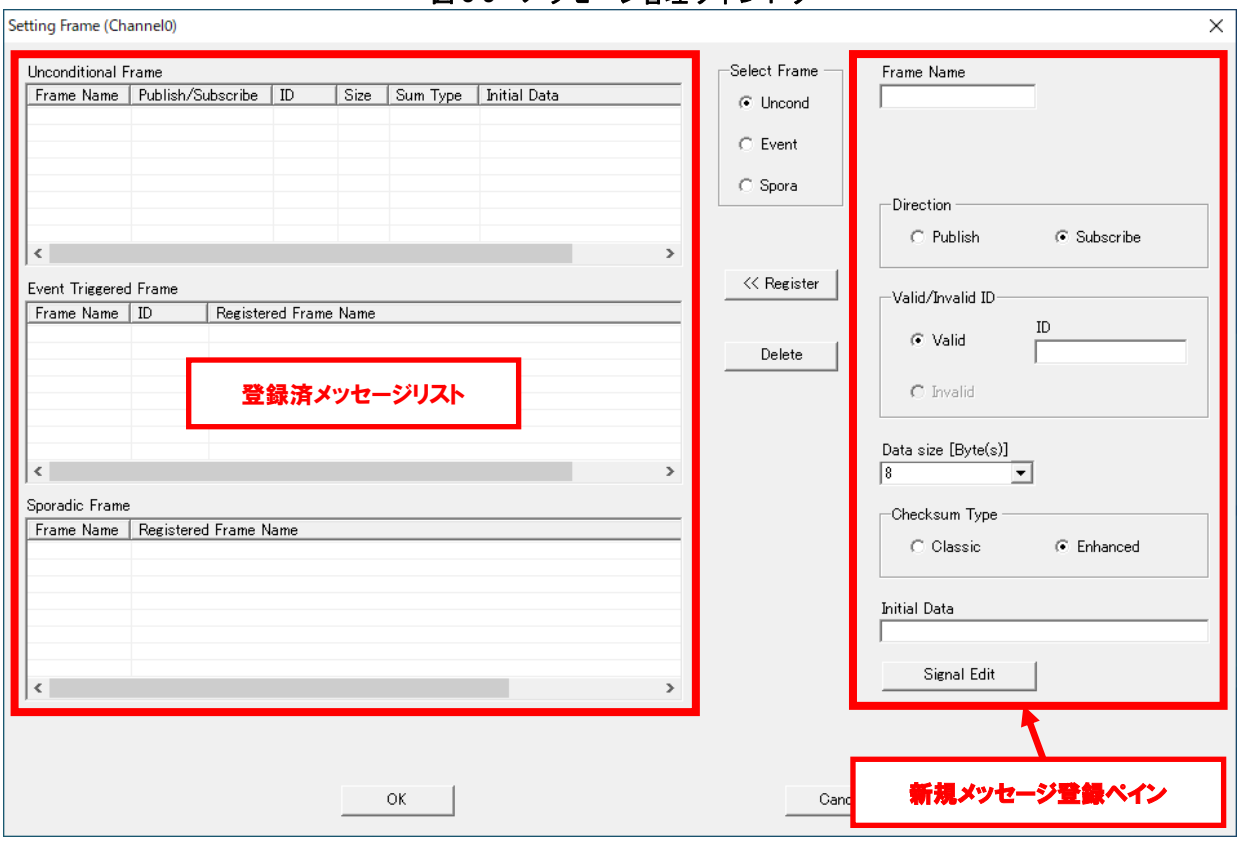
スレーブの場合、ボー・レート設定で”Auto”を選択可能です。”Auto”を選択した場合はマスターから送信された LIN のヘッダから自身のボー・レートを自動設定します。

【注意事項】

開発環境のプロジェクト・ファイルにボー・レートの検出方法(“Auto”、”Manual”)をコンパイル・オプション(“__LIN_HW_AUTO_BR__”など)として記述する必要があります。詳しくはコンフィギュレータによって出力される readme.txt を参照してください。

6. 2. 6　メッセージ管理

図 6-9　メッセージ管理ウィンドウ



チャネル毎のメッセージ(LIN フレーム)の管理を行います。

“Sporadic Frame”はマスターチャネルでのみ選択可能です。

“Valid/Invalid ID”には LIN のフレーム ID を設定します(0x00～0x3B)。パリティは不要です。スレーブチャネルの場合のみ”Invalid”を選択可能で、その場合はフレーム ID として”0xFF”が設定されます。

“Initial Data”にはメッセージの初期データをカンマで区切って設定します。例として、8 バイトデータの全てに 0 を設定する場合、”0,0,0,0,0,0,0,0”と記述します。

新規メッセージ登録時、右側の「新規メッセージ登録ペイン」内に登録したいメッセージの情報を入力し、”Select Frame”ラジオボタンでフレーム種別を選択して”Register”を押すと、左側のリストに登録されます。

左側の「登録済メッセージリスト」内のメッセージを選択した状態で”Delete”を押すとメッセージを削除可能です。

登録済みのメッセージを修正する場合は以下の手順を行い、開いた各ウィンドウ内で修正を行います。

“Unconditional Frame”内のメッセージをダブルクリックすると、無条件フレーム設定ウィンドウが開きます。

“Event Triggered Frame”内のメッセージをダブルクリックすると、イベント・トリガ・フレーム設定ウィンドウが開きます。

“Sporadic Frame”内のメッセージをダブルクリックすると、スポラディック・フレーム設定ウィンドウが開きます。

【注意事項】

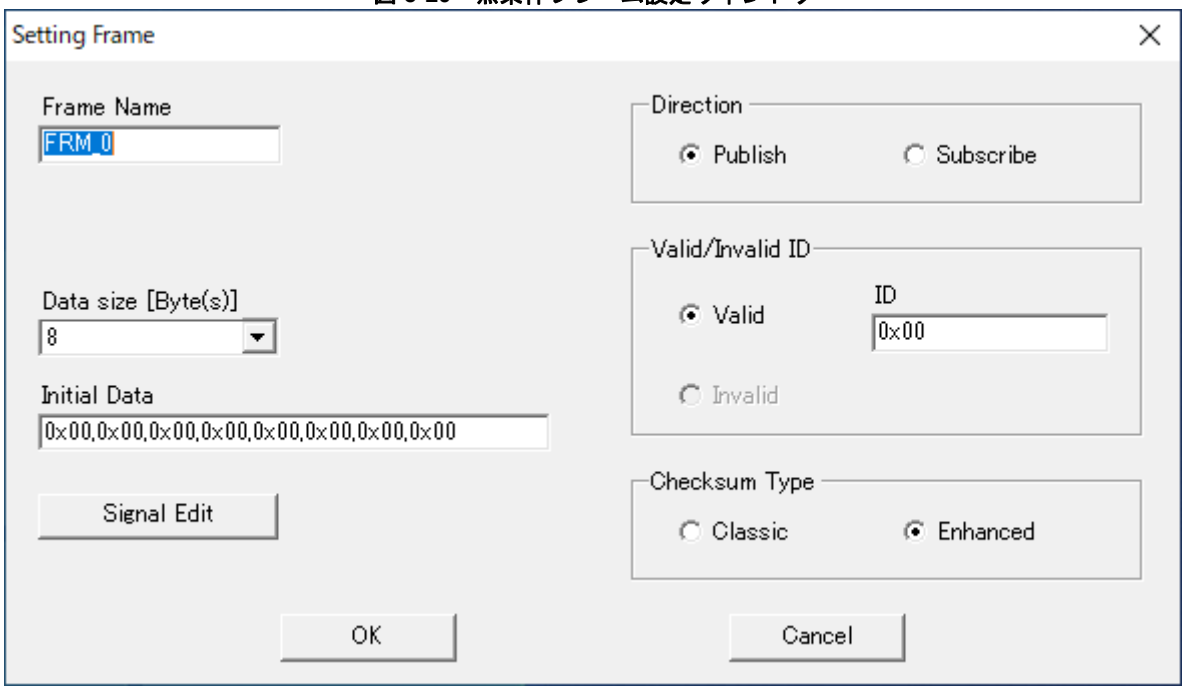
“Frame Name”の文字列は 256 文字以内にしてください。使用可能文字は半角英数字と”_”です。ただし、文字列先頭に数字は使えません。

“ID”の設定可能範囲は 0x00～0x3B です。範囲外の値を指定しないでください。

“Initial Data”は”Size”で指定した数だけ定義してください。

（1） 無条件フレーム設定

図 6-10 無条件フレーム設定ウィンドウ



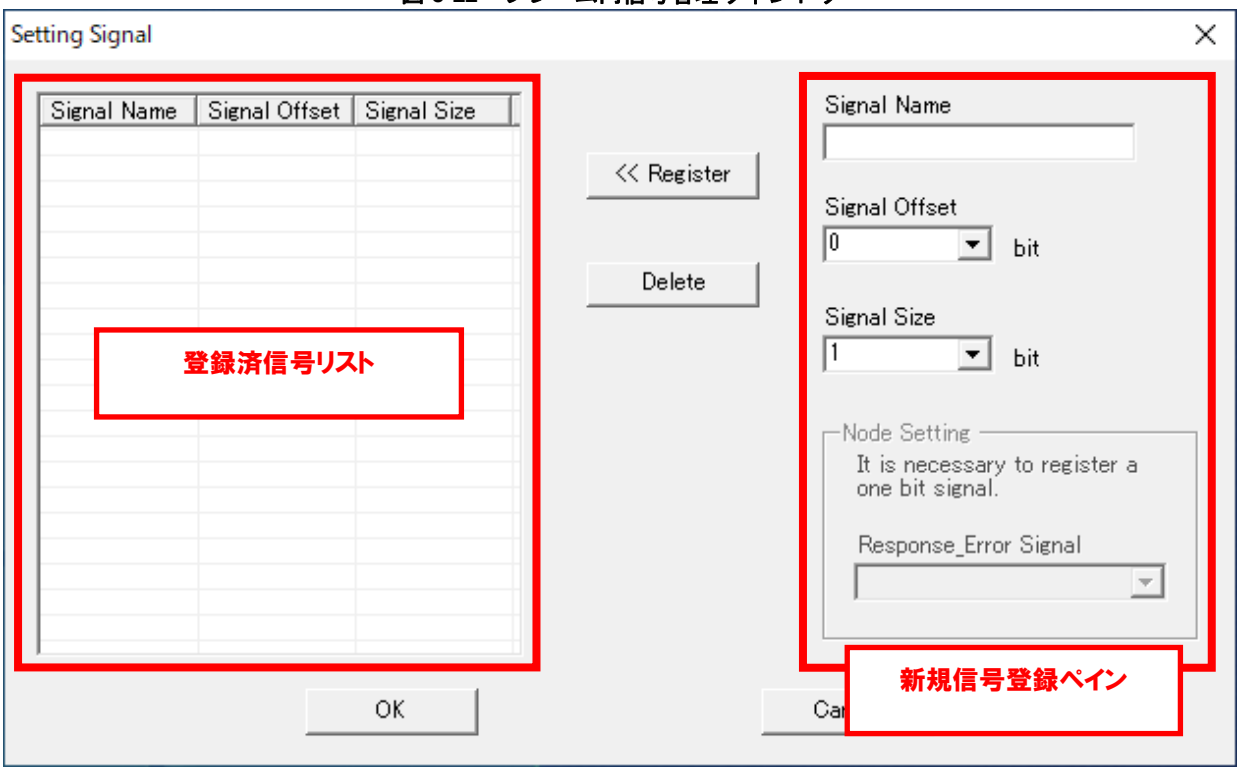
「登録済メッセージリスト」の”Unconditional Frame”内のメッセージをダブルクリックすると開きます。

設定後、”OK”を押すと設定がメッセージに反映されます。

“Signal Edit”を押すと、メッセージ内の信号を管理するウィンドウ(シグナル管理ウィンドウ)が開きます。

（2） フレーム内信号管理

図 6-11 フレーム内信号管理ウィンドウ



メッセージ内の信号を管理します。

右側の「新規信号登録ペイン」内の項目を設定し、”Register”を押すと新たに信号を登録可能です。

「登録済信号リスト」内の信号を選択した状態で”Delete”を押すと信号を削除することが可能です。

「登録済信号リスト」内の信号をダブルクリックすると「シグナル設定ウィンドウ」が開きます。

スレーブチャネルの場合、チャネルにつき”Response_Error_Signal”が 1 つ必要です。”Response_Error_Signal”には送信メッセージ内の 1 ビット長の信号のみ割り当てることが可能です。Direction が Publish の無条件フレームに対して Signal Size が 1 の信号を設定することにより、”Response_Error_Signal”が選択可能になります。

【注意事項】

“Signal Name”の文字列は 256 文字以内にしてください。使用可能文字は半角英数字と”_”です。ただし、文字列先頭に数字は使えません。

同一フレームにシグナルを複数登録する場合は、シグナル同士の定義領域が重ならないようにしてください。

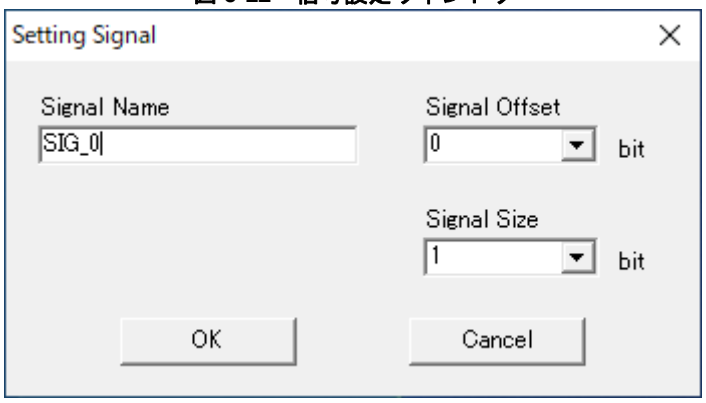
3 バイト以上にまたがるシグナルを定義しないでください。

フレームの範囲外にシグナルを定義しないでください。

サイズが 16 ビット以上のときは、バイト・アレイ・シグナルとして定義してください。

（3） 信号設定

図 6-12 信号設定ウィンドウ

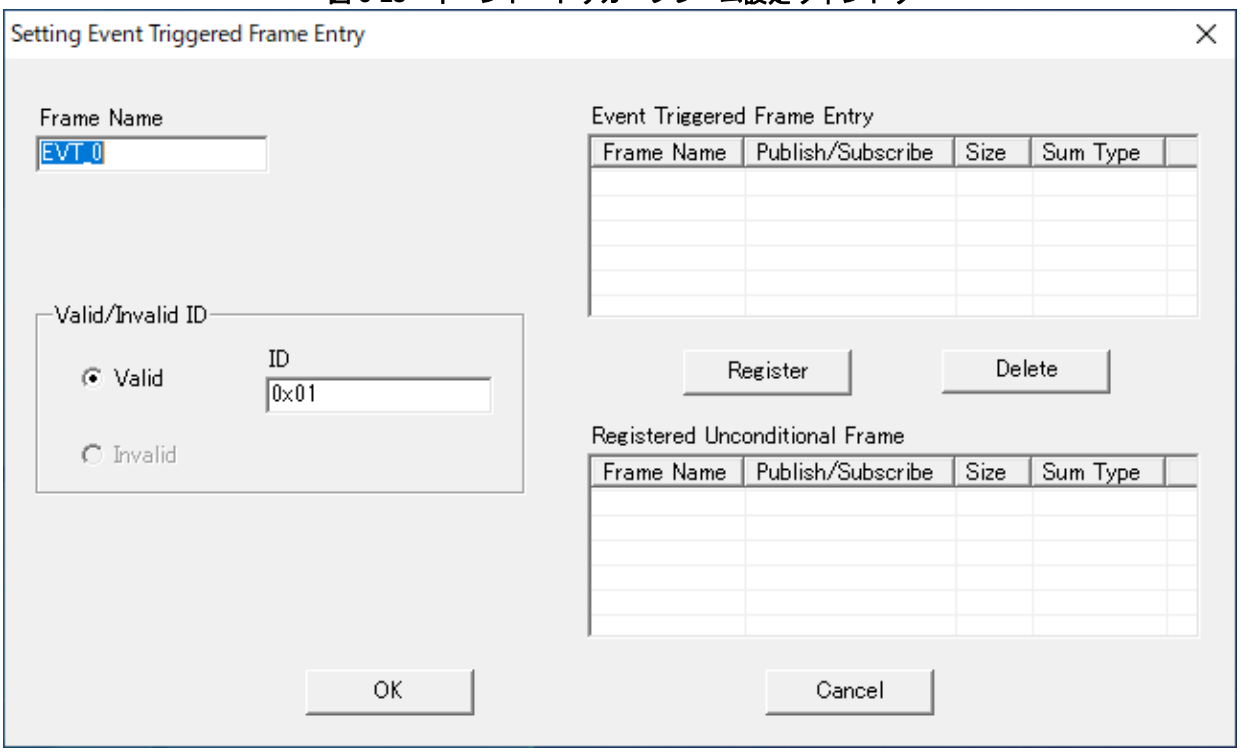


登録済信号の設定を変更します。

設定完了後、”OK”を押し、”Setting Signal”ウィンドウ、”Setting Frame”ウィンドウで”OK”を押すと設定が反映されます。

（4） イベント・トリガ・フレーム設定

図 6-13 イベント・トリガ・フレーム設定ウィンドウ



「登録済メッセージリスト」の”Event Triggered Frame”内のメッセージをダブルクリックすると開きます。

“Registered Unconditional Frame”リスト内の無条件フレームを選択し、”Register”を押すと、イベント・トリガ・フレームに関連付けることが可能です。

”Valid/Invalid ID”の意味については無条件フレームと同様です。

“OK”を押し、”Setting Frame”ウィンドウで”OK”を押すと設定が反映されます。

【注意事項】

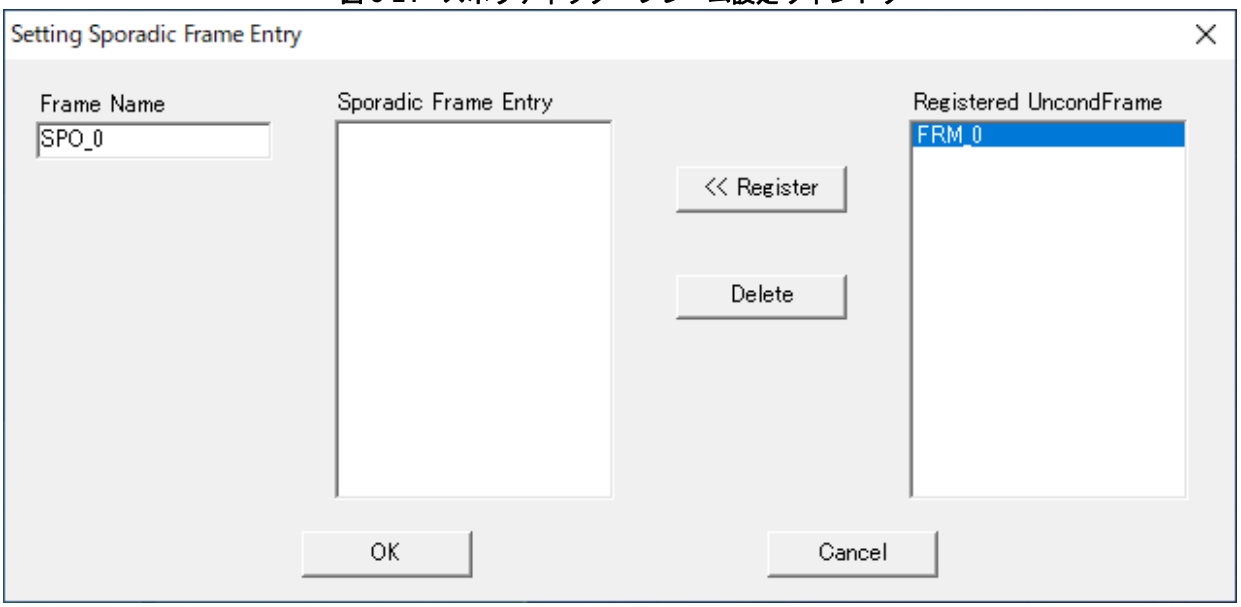
“Frame Name”の文字列は 256 文字以内にしてください。使用可能文字は半角英数字と”_”です。ただし、文字列先頭に数字は使えません。

“ID”の設定可能範囲は 0x00～0x3B です。範囲外の値を指定しないでください。

複数の無条件フレームを同一イベント・トリガ・フレームに登録する場合は、登録する各々の無条件フレームの通信方向、データ・サイズ、チェックサム・タイプを一致させてください。

（5） スポラディック・フレーム設定

図 6-14 スポラディック・フレーム設定ウィンドウ



「登録済メッセージリスト」の”Sporadic Frame”内のメッセージをダブルクリックすると開きます。

“Registered UncondFrame”リスト内の無条件フレームを選択し、”Register”を押すとスポラディック・フレームに関連付けることができます。

“OK”を押し、”Setting Frame”ウィンドウで”OK”を押すと設定が反映されます。

【注意事項】

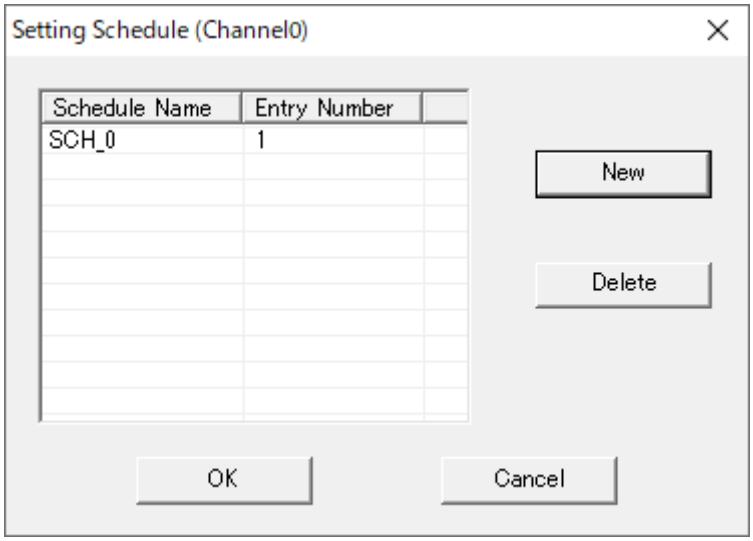
“Frame Name”の文字列は 256 文字以内にしてください。使用可能文字は半角英数字と”_”です。ただし、文字列先頭に数字は使えません。

6. 2. 7　スケジュール管理

LIN フレームの送受信タイミングをまとめたテーブルをスケジュール・テーブルと呼びます。

（1）　スケジュール・テーブル管理

図 6-15　スケジュール・テーブル管理ウィンドウ



スケジュール・テーブルの管理を行います。マスターチャネルでのみ使用します。

“New”、もしくはリスト内のスケジュールを押すとスケジュール・テーブル設定ウィンドウが開きます。

“OK”を押すと設定が反映されます。

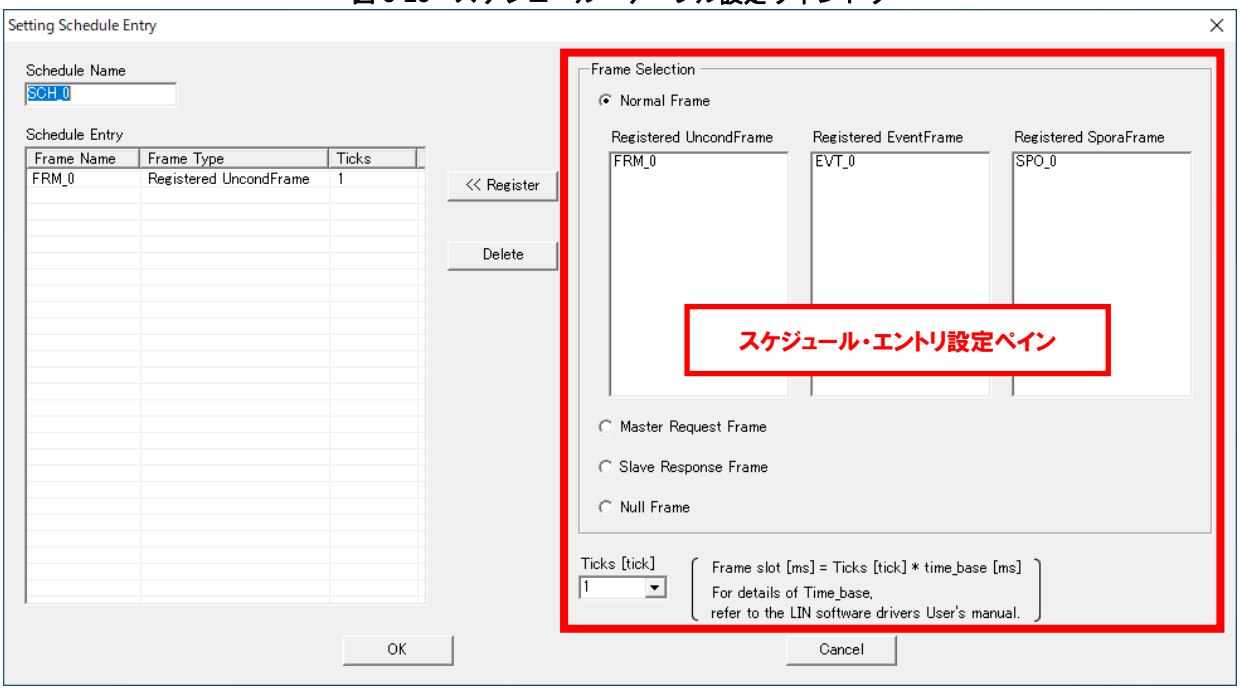
【注意事項】

スケジュール・テーブルの定義可能範囲は 1～255 です。

スケジュールが登録されていない場合でも”OK”を押すことができ、LIN ドライバのソース・コードを出力することが可能です。

（2） スケジュール・テーブル設定

図 6-16 スケジュール・テーブル設定ウィンドウ



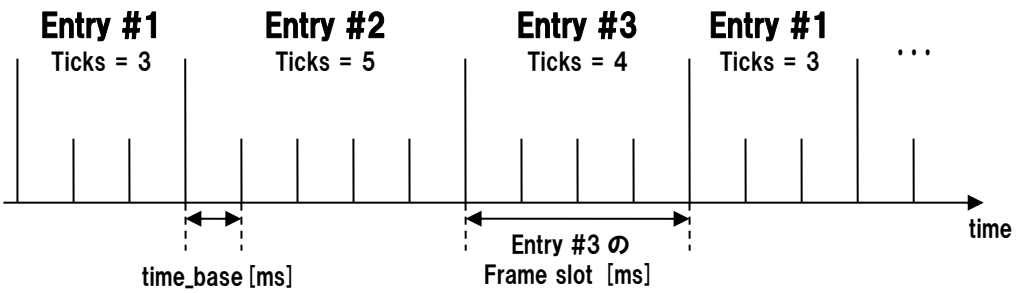
LIN ヘッダ送信スケジュールの設定を行います。マスターチャネルでのみ使用します。

“Schedule Name”にスケジュール名を記述します。スケジュール名は 1 つのチャネル内で重複できません。

スケジュール・エントリ設定ペイン内から、スケジュールに追加するフレームと Tick を選択し”Register”を押すとスケジュール・エントリに追加されます。

例として Entry#1:Tick=3、Entry#2:Tick=5、Entry#3:Tick=4 を設定した場合のスケジュールの動きを以下に示します。

図6-17 スケジュール・テーブル（Entry#1、#2、#3）の動き

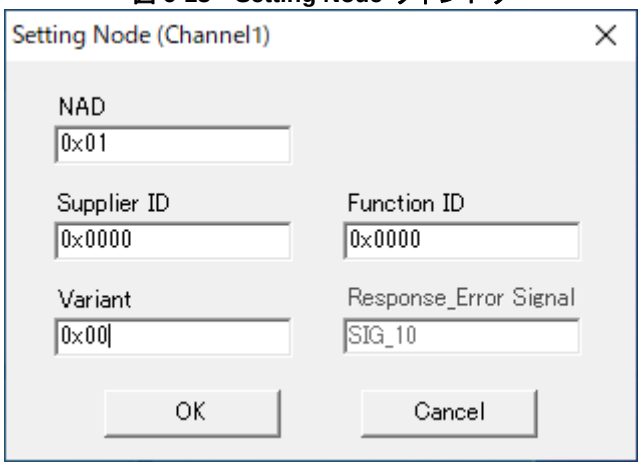


スケジュール・エントリ内のエントリを選択し、”Delete”を押すとエントリを削除できます。

“OK”を押すと設定が反映されます。

6. 2. 8　ノード設定

図 6-18　Setting Node ウィンドウ



LIN スレーブのノード情報を設定します。

“NAD”にはノード・アドレス(1～127)を設定します。

“Supplier ID”、”Function ID”、”Variant”にはそれぞれ製品の ID を設定します。

“Response_Error Signal”には、”Setting Signal”ウィンドウで設定された”Response_Error Signal”信号名が表示されます。この画面での変更はできません。

“OK”を押すと設定が反映されます。

【注意事項】

NADの設定可能範囲は 1 ～255（0x01～0xFF）です。範囲外の値を指定しないでください。

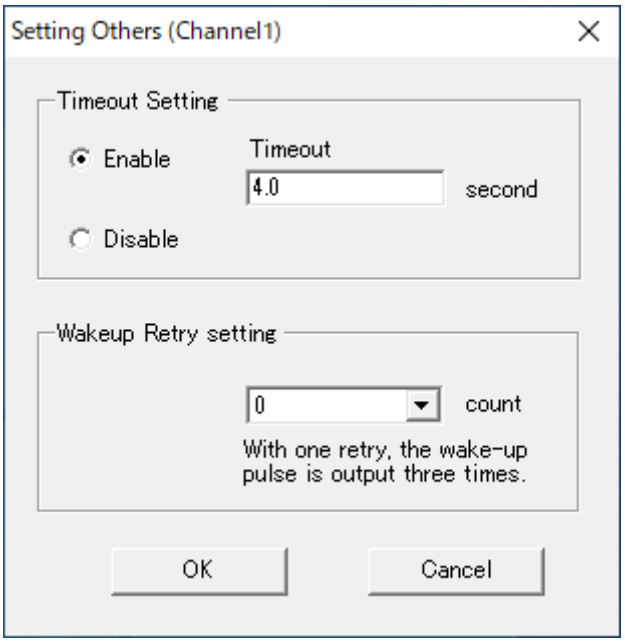
Supplier IDの設定可能範囲は0～65535（0x0000～0xFFFF）です。範囲外の値を指定しないでください。

Function IDの設定可能範囲は0～65535（0x0000～0xFFFF）です。範囲外の値を指定しないでください。

Valiantの設定可能範囲は0～255（0x00～0xFF）です。範囲外の値を指定しないでください。

6. 2. 9　その他設定

図 6-19　Setting Others ウィンドウ



LIN スレーブのバス・タイムアウト、およびウェイクアップ・パルス出力のリトライ回数を設定します。

バス・タイムアウトは、“Enable”を選択した場合、タイムアウト時間を設定します。LIN バスがここで設定した時間 In-Active だった場合、LIN スレーブはスリープ・モードに入ります。

“Disable”を選択した場合、LIN ドライバはバス・タイムアウトによるスリープを行いません。

ウェイクアップ・パルス出力のリトライ回数は 0～80 の範囲で設定することが可能です。

“OK”を押すと設定が反映されます。

【注意事項】

Timeoutの設定可能範囲は0.1～18.0[sec]です。範囲外の値を指定しないでください。

6. 2. 10　設定ファイルの保存、読出し

メイン・ウィンドウ上部の”Save”ボタン押下、もしくは”File”メニュー内の”Save”、”Save As”を選択すると、LIN コンフィギュレータの設定内容を XML 形式で保存可能です。

また、上部の”Open”ボタン押下、もしくは”File”メニュー内の”Open”を選択すると、保存してある設定ファイルを読み込むことができます。

6. 2. 11　ソース・コード出力

メイン・ウィンドウ上部の”Go”ボタン押下、もしくは”Tool”メニュー内の”Generate Source Code”を選択すると LIN ドライバのソース・コード、開発環境プロジェクト・ファイルが出力されます。

各ファイルの出力先は以下のようになります。
(R7F124FPJ3xNP でチャネル 0 とチャネル 1 を選択した場合の例です)

図 6-20　ソース・コード出力先

```
(ユーザ指定フォルダ) /r_lin_drv
 ├ /Channel0
 │  ├ /conf
 │  │  ├ conflin_0.c
 │  │  └ conflin_0.h
 │  │
 │  ├ /liblin2
 │  │  └ (開発環境プロジェクト・ファイル群)
 │  │
 │  └ /libsrc
 │     └ (LIN ドライバ・ソースコード)
 │
 └ /Channel1
    ├ /conf
    │  ├ conflin_1.c
    │  └ conflin_1.h
    │
    ├ /liblin2
    │  └ (開発環境プロジェクト・ファイル群)
    │
    └ /libsrc
       └ (LIN ドライバ・ソースコード)
```

【補足事項】

コンフィギュレータは CC-RL 用と IAR コンパイラ用のプロジェクト・ファイルを出力します

6.3 エラー・メッセージ一覧

表 6-3 エラー・メッセージ一覧

エラー・メッセージ	原因
「XXX is not specified」 (XXX は項目の見出し)	項目 XXX に値が入力されていない状態で OK, Register が押下された。
「XXX is not numerical number」 (XXX は項目の見出し)	数値入力欄で数値文字列以外が入力された状態で OK, Register が押下された。
「XXX overs the maximum limit」 (XXX は項目の見出し)	数値入力欄で最大値を上回る数値が入力された状態で OK, Register が押下された。
「XXX is smaller than the minimum limit」 (XXX は項目の見出し)	数値入力欄で最小値を下回る数値が入力された状態で OK, Register が押下された。
「The input value of XXX is illegal」 (XXX は項目の見出し)	数値指定用コンボボックスでリストにない数値文字列が入力された状態で OK, Register が押下された。(キーボードから文字列入力が可能な場合)
	名称入力欄で C 言語の識別子文字パターンに一致しない文字列が入力された状態で OK, Register が押下された。
「Can not open the device entry file」	コンフィギュレータ付属の XML ファイルの読み込みに失敗した。
「Can not open the specified file」	コンフィギュレータのプロジェクト･ファイル、またはスマート・コンフィグレータが出力した LIN 関連ヘッダ・ファイルの読み込みに失敗した。
	コンフィギュレータのプロジェクト･ファイルの書き込みに失敗した。
「Illegal file format of the device entry file」	コンフィギュレータ付属の XML ファイルの読み込みで、XML 記述が正しく解釈できなかった。
「Illegal file format」	コンフィギュレータのプロジェクト･ファイル、またはスマート・コンフィグレータが出力した LIN 関連ヘッダ・ファイルの読み込みで、XML 記述が正しく解釈できなかった。
「Can not make a config file」	コンフィグ・ファイル生成に失敗した。
「Can not make a library file」	ライブラリの出力に失敗した。
「Baud rate is not specified」	ボー・レート設定画面での設定が行われていない状態でコンフィグ・ファイル生成が実行された。
「One or more unconditional frames are needed」	登録されている無条件フレームがない状態でコンフィグ・ファイル生成が実行された。
「Signal is not specified」	登録されているシグナルがない状態でコンフィグ・ファイル生成が実行された。
「Schedule is not specified」 (Master Only)	スケジュール設定画面での設定が行われる前にコンフィグ・ファイル生成が実行された。
「Node is not found」 (Slave Only)	ノード設定画面での設定が行われる前にコンフィグ・ファイル生成が実行された。
「Response_Error Signal is not specified」 (Slave Only)	Response_Error Signal が設定される前にコンフィグ・ファイル生成が実行された。
「The extra information is not specified」 (Slave Only)	その他設定画面での設定が行われる前にコンフィグ・ファイル生成が実行された。
「There is a frame that Valid ID is not specified」 (Master Only)	Valid ID が設定されていないフレームがある状態でコンフィグ・ファイル生成が実行された。
「Frame Name is specified redundantly」	Frame Name に他データと同じ名称が入力された状態で OK、Register が押下された。
「The input value of Initial Data is illegal」	Initial Data の数値入力個数が指定 Size に一致しない状態で OK 、Register が押下された。

エラー・メッセージ	原因
「The input value of Direction is illegal」	関連付けられた無条件フレームを複数持つイベント･トリガ･フレームに関連付けられた無条件フレームの更新中に Size を変更された状態で OK が押下された。
「Can not register Unconditional Frame any more」	登録エントリが最大数に達している状態で New、Register が押下された。
「Can not register Event Triggered Frame any more」	
「Can not register Sporadic Frame any more」	
「Can not register Signal any more」	
「Can not register Event Triggered Frame Entry any more」	
「Can not register Sporadic Frame Entry any more」	
「Can not register Schedule any more」	
「Can not register Schedule Entry any more」	
「Can not update the Frame Name」	イベント・トリガ・フレームまたはスポラディック・フレームに関連付けられた無条件フレームか、スケジュール・テーブルのエントリに含まれる無条件フレームの更新中に Frame Name を変更された状態で OK が押下された。
	スケジュール・テーブルのエントリに含まれるイベント・トリガ・フレームの更新中に Frame Name を変更された状態で OK が押下された。
	スケジュール・テーブルのエントリに含まれるスポラディック・フレームの更新中に Frame Name を変更された状態で OK が押下された。
「Can not update the Size」	関連付けられた無条件フレームを複数持つイベント・トリガ・フレームに関連付けられた無条件フレームの更新中に Size を変更された状態で OK が押下された。
「Can not update the Direction」	以下のいずれかに該当するフレームの更新中に Direction が変更された状態で OK が押下された。 a)Master かつイベント・トリガ・フレームの関連するフレームに登録されている b)Slave かつイベント・トリガ・フレームの関連するフレームに登録され、同一のイベント・トリガ・フレームに関連付けられている無条件フレームが他に存在する
「Can not update the Checksum Type」	関連付けられた無条件フレームを複数持つイベント・トリガ・フレームに関連付けられた無条件フレームの更新中に Checksum Type を変更された状態で OK が押下された。
「The input of Direction is illegal」	Response_Error に割り当てられたシグナルを持っている状態で Direction を Subscribe にして OK が押下された。
「The input of Event Triggered Frame Entry is illegal」	現在リストされているエントリと Direction、Size、Checksum Type が一致しない無条件フレームが選択された状態で Register が押下された。
	現在リストされているエントリに不正なデータがある状態で OK, Register が押下された。
「The input of Sporadic Frame Entry is illegal」	現在リストされているエントリに不正なデータがある状態で OK が押下された。
「Significant digit of a Timeout is one place of decimals」	Timeout に小数第一位より後の入力がある状態で OK が押下された。
「Schedule Name is specified redundantly」	Schedule Name に他データと同じ名称が入力された状態で OK が押下された。
「Data of Schedule Entry is not adjusted」	現在リストされているエントリに不正なデータ（Frame Name に対応する Frame Type に誤りがある）がある状態で OK, Register が押下された。

エラー・メッセージ	原因
「Normal Frame is not selected」	Normal Frame のスケジュール・エントリ登録の際、対象フレームが選択されていない状態で Register が押下された。
「Signal Name is specified redundantly」	Signal Name に他データと同じ名称が入力された状態で Register が押下された。
「The input of Signal Offset is illegal」	Signal Size が 16bit を超えているときに Signal Offset に８の倍数以外が入力された状態で OK、Register が押下された。
「The input of Signal range is illegal」	以下の状態で Register が押下された。 a)シグナルの範囲が指定データ・サイズに包含されない b)シグナルの範囲が３バイトにまたがっている c)シグナルの範囲が他のシグナルと重なっている
	指定データ・サイズに包含されないシグナルがある状態で OK が押下された。
「Can not update the Signal Name」	エラーシグナルとして登録されているシグナルの更新中に Signal Name を変更された状態で OK が押下された。
「Can not update the Signal Size」	エラーシグナルとして登録されているシグナルの更新中に Signal Size を 1 以外に変更された状態で OK が押下された。

6.4 ワーニング・メッセージ一覧

表 6-4 ワーニング・メッセージ一覧

ワーニング・メッセージ	原因
「Do you want to create a new project without saving the current project?」	ファイルへ保存していない設定がある状態で，[File]-[New]（プロジェクトの新規作成）が行われた。
「Do you want to save the project file?」	ファイルへ保存していない設定がある状態で，[File]-[Exit]または，ウィンドウ右上の×ボタンの押下が行われた。
「Do you want to load the project without saving the current project?」	ファイルへ保存していない設定がある状態で，[File]-[Open]（既存プロジェクトの読み込み）が行われた。
「conflin.h already exists. Do you want to overwrite it?」	指定されたコンフィグ・ファイル生成フォルダにすでにコンフィグ･ファイルが存在する。

第7章 LIN2.1 ソフトウェア・ドライバ概要

7.1 シグナルタイプ 【マスター/スレーブ】

LIN2.1 ソフトウェア・ドライバでは、1～16 ビット幅のスカラ・シグナルと、1～8 バイト幅のバイト･アレイ・シグナルの 2 種類を扱うことができます。シグナルは conflin_x.c を編集することにより定義可能です。

1 フレーム中に複数のシグナルを任意の場所に定義することができます。ただし、2 つ以上のバイト境界を越えるスカラ・シグナルを定義することはできません。

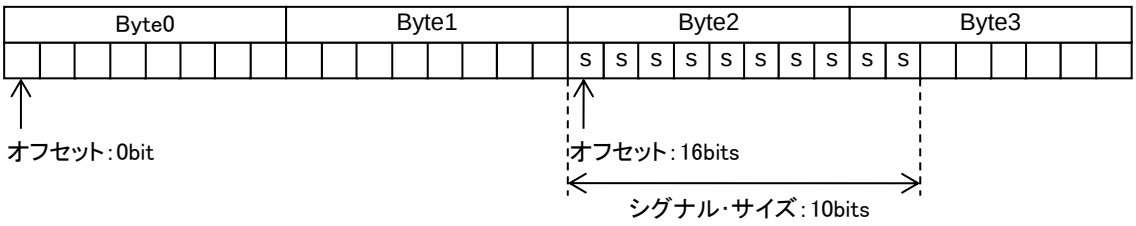
ユーザ･アプリケーションからシグナルへのアクセスは、LIN2.1 ソフトウェア・ドライバ関数をコールすることにより可能です。

例：4 バイト長フレームに 10 ビット・シグナル“S”を含む場合（メッセージ･バッファ 0）

条件：

フレームサイズ	：4 バイト
シグナル初期値	：0x03FF
オフセット	：16 ビット
シグナル・サイズ	：10 ビット

図7-1 メッセージ・データへのシグナル配置例



7.2 フレーム構成 【マスター/スレーブ】

LIN 通信におけるフレーム構成は、以下の通りです。

図7-2 フレーム構成

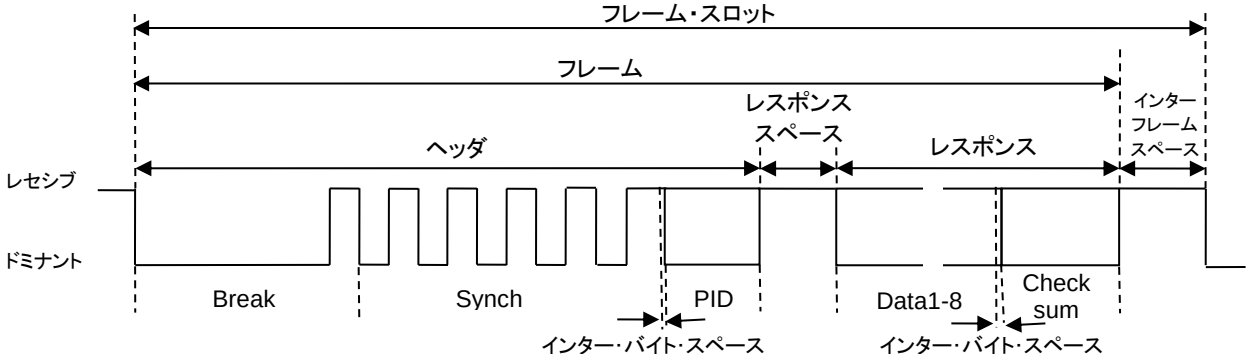


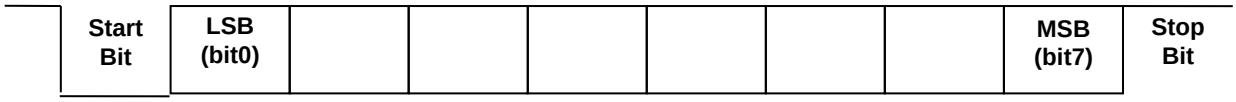
表 7-1 フレーム構成要素一覧

用語	意味
フレーム・スロット	マスターによって管理される､1回のフレーム転送に割り当てられる通信時間
フレーム	LIN通信の通信単位
ヘッダ	BreakからPIDまでのマスターからの送信データ
レスポンス・スペース	ヘッダ送信完了から、レスポンス送信開始までの時間
レスポンス	ヘッダ内に含まれるPIDに従って、スレーブ又はマスターから送信されるデータ
インター・フレーム・スペース	1通信終了後(Checksumの転送完了)から、次回通信開始（Breakの立ち下がりエッジ発生）までの時間。0以上である必要があります。
インター･バイト･スペース	Stop Bitから次のStart Bitまでの時間（図5-3参照）。 0以上である必要があります。
Break	13ビット以上のLowパルス。スレーブでは11ビット以上であればBreakとみなします。
Synch	0x55データ、スレーブがボー・レート検出のために利用します。
PID	フレームを識別する為のパリティ付きフレームID。上位2ビットはパリティ、下位6ビットがフレームIDとなっています。
Data1-8	1～8バイトのデータ。初期データ値は、コンフィグ・ファイルにて指定します。
Checksum	キャリーオーバ付きの8ビットチェックサムの反転値。クラシック・チェックサムは全Data、拡張チェックサムはPID＋全Dataを用いてチェックサムを算出します。 診断フレームではクラシック・チェックサムが使用され、その他の通信では拡張チェックサムを使用することができます。

7.2.1 バイト・フィールド

Break以外は以下のバイト･フィールド･フォーマットで構成されています。

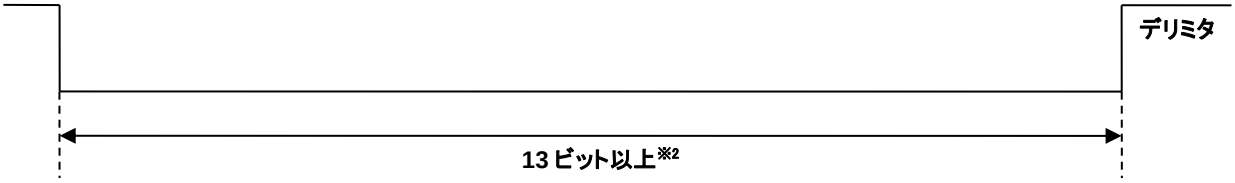
図7-3 バイト・フィールド・フォーマット



7.2.2 Break フィールド

Breakフィールドは13ビット以上※1のLowパルスと、1ビット以上のHighパルス（デリミタ）で構成されます。
LIN2.1スレーブでは、11ビット以上のLowパルスをBreakとして認識します。

図7-4 Breakフォーマット



※1：LIN2.1 マスター・ドライバでは、13～20 ビットまで設定可能です。
※2：最大値はヘッダ全体としての最大値（最小値の 1.4 倍）に依存します。

7.2.3 フレーム長

フレーム転送にかかる時間の公称値は、転送されるビット数に一致します（だたし、レスポンス･スペース、インター・バイト・スペースを含まない値）。
以下にフレーム長の計算式を示します。

$$T_{Header_Nominal} = 34 \times T_{Bit}$$
$$T_{Response_Nominal} = 10 \times (N_{Data} + 1) \times T_{Bit}$$
$$T_{Frame_Nominal} = T_{Header_Nominal} + T_{Response_Nominal}$$

※ T_{Bit} ＝1 ビット時間、 N_{Data} ＝データバイト数

LIN2.1 Spec では、通常の転送時間より最大 40%の誤差が許容されています。
以下に最大フレーム長の計算式を示します。

$$T_{Header_Max} = 1.4 \times T_{Header_Nominal}$$
$$T_{Response_Max} = 1.4 \times T_{Response_Nominal}$$
$$T_{Frame_Max} = T_{Header_Max} + T_{Response_Max}$$

7.3　フレーム転送　【マスター/スレーブ】

LIN2.1 Spec で定義されているフレームは以下の通りです。

表 7-2　フレーム種類一覧

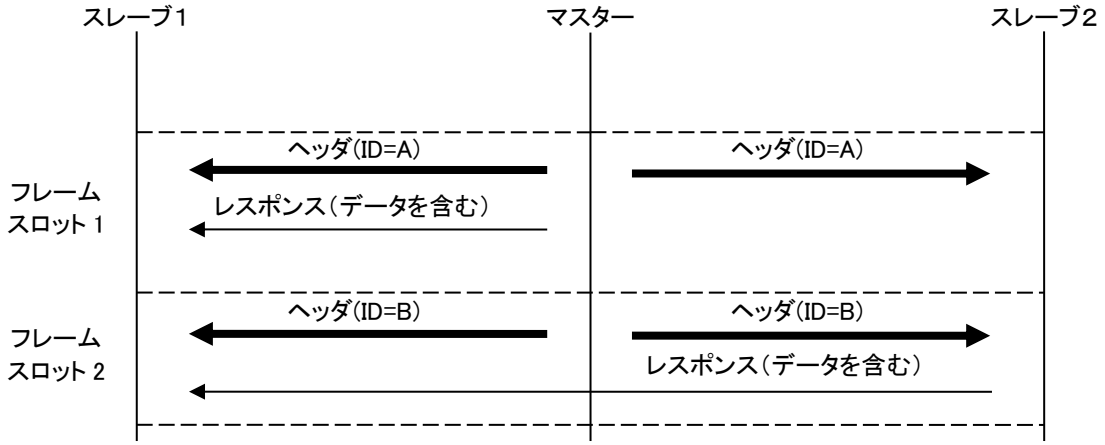
フレーム名	説明
無条件フレーム	通常の送受信に用いるフレーム
イベント･トリガ･フレーム	更新のあるスレーブからのみ、情報を取得するフレーム
スポラディック･フレーム	スケジュール外の情報を扱うフレーム
診断フレーム	ノード･コンフィギュレーションのために特定の情報を送受信するフレーム マスター･リクエスト･フレームとスレーブ･レスポンス･フレームがあります

以下に、各フレームの転送例を示します。

7.3.1　無条件フレーム転送

マスターからヘッダを送信することにより、対象となるマスター・ノードまたはスレーブ・ノードから、レスポンスが送信されます。送信されたレスポンスは、受信対象となるマスターまたはスレーブが受信します。

図7-5　データ送受信例



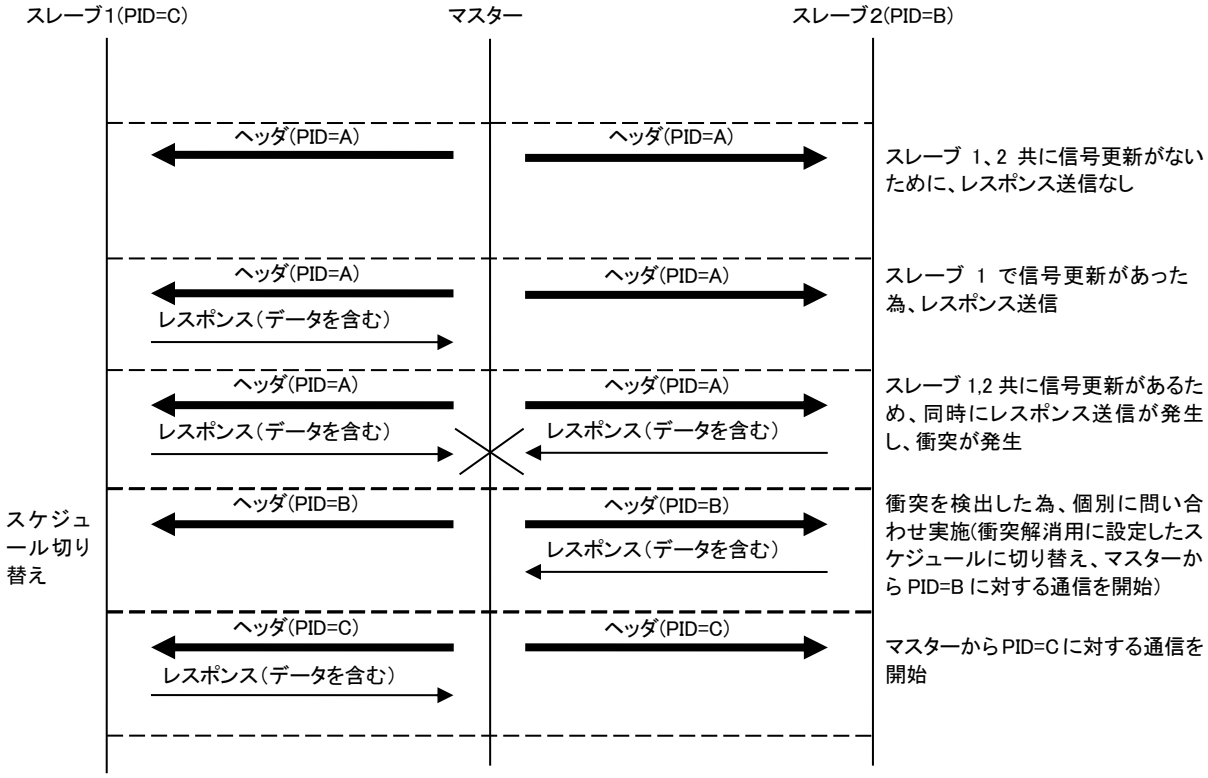
7.3.2 イベント・トリガ・フレーム転送

マスターは、イベント・トリガ・フレームを用いて、イベント･トリガ･フレームに関連付けられた無条件フレームのレスポンスを受信します（レスポンスの送信はサポートしません）。スレーブはイベント･トリガ･フレームのヘッダを受信すると、シグナル更新があった場合のみ、レスポンスを送信します。

もし、どのスレーブもシグナル更新がなかった場合、レスポンスは空となります。また、2つ以上のノードが同時にレスポンスを送信した場合、衝突が発生します。この場合、マスター・ノードは衝突を解消する必要があります。

イベント・トリガ・フレームをPID=A、関連する無条件フレームをPID=B、PID=Cとすると、次のような通信が実施されます。

図7-6 イベント･トリガ･フレームによるデータ送受信例



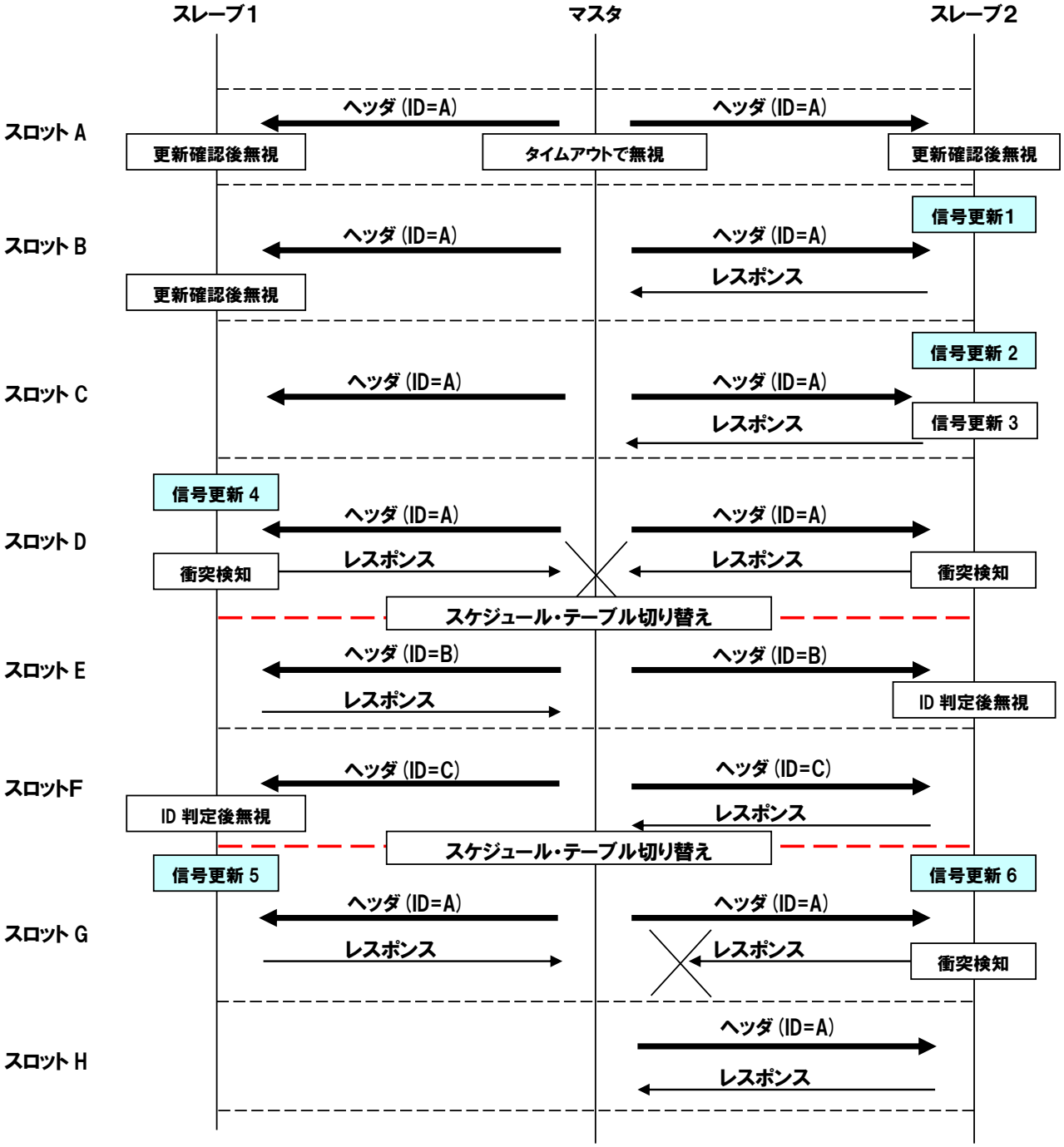
※注意：

1. イベント･トリガ･フレーム受信においてエラーを検出した場合は、衝突として扱われるため、受信完了やエラーとなりません。
2. イベント・トリガ・フレーム ID は、0x00～0x3B（パリティを含まない値）でなければなりません。
3. レスポンスの最初の 1 バイトで関連する無条件フレームの PID を転送するため、ここにシグナルを割り当てないでください。従って、先頭 1 バイトを除いた最大 7 バイト分の領域にシグナルを割り当てることが可能です。
4. イベント・トリガ・フレームに関連付ける無条件フレームは同じチェックサム・モデルにしてください。
5. イベント・トリガ・フレームに関連付ける無条件フレームのデータ長は同じにしてください。
6. イベント・トリガ・フレームに関連付ける無条件フレームは、すべて異なるノードから送信される必

要があります。

7．イベント・トリガ・フレームとそのフレームに関連付けられた無条件フレームは、同じスケジュール・テーブルに存在できません。

LIN 2.1ソフトウェア・ドライバでは、衝突回避のためにマスターで衝突を解消するための調整を実施します。



スロット A：マスターから両スレーブに対して問い合わせ（ID=A[0x00-0x3B]、イベント・トリガ用 ID)、レスポンスなしの場合

スロット B：マスターから両スレーブに対して問い合わせ（ID=A[0x00-0x3B]、イベント・トリガ用 ID)、スレーブ２からマスターへデータ送信（更新値１）

スロットC：マスターから両スレーブに対して問い合わせ（ID=A[0x00-0x3B]、イベント・トリガ用 ID)、スレーブ２からマスターへデータ送信（更新値 2)、更新と重なる時

スロットD：マスターから両スレーブにたいして問い合わせ（ID=A[0x00-0x3B]、イベント・トリガ用 ID)、両

スレーブからマスターへデータ送信があり、両スレーブともに衝突状態となり、マスターが衝突検出できる場合。

自動的に衝突解決スケジュールへ切り替わり。

スロットＥ：衝突検知後、スレーブ１からマスター（ID=B[0x00-0x3B]、無条件フレーム用）へデータ送信（更新値 4）

スロットＦ：衝突検知後、スレーブ２からマスター（ID=C[0x00-0x3B]、無条件フレーム用）へデータ送信（更新値 3）。

すべての衝突が解決されたら自動的に元のスケジュールに切り替わり。

スロット G:マスターから両スレーブにたいして問い合わせを実施（ID=A[0x00-0x3B]､イベント･トリガ用 ID）。両スレーブからマスターへデータ送信があるが、スレーブ２内部ではエラー発生を検出するが、LIN ライン上には、更新値 5 がそのまま送信され、マスターは衝突検出でなく、スレーブ 1 から正常にデータ送信されたと解釈される。（※１：フレーム例参照）

スロット H : スレーブ２からマスター（ID=C[0x00-0x3B]、無条件フレーム用）へデータ送信（更新値６）。前回はエラーでデータを送信していない為、更新値６が送信される。

参考：

※１マスターが衝突検出できない場合のフレーム例

ID	Data	Sum
0x80	**0x00**	**0x7F**

&

ID	Data	Sum
0xc1	0xbe	0x7F

=

ID	Data	Sum
0x80	**0x00**	**0x7F**

※Sum は従来のチェックサムです。

【注意】

1. イベント・トリガ・フレーム送受信においてエラーを検出した場合は、衝突として扱われる為、エラーとなりません。また、送信成功ともしません。
2. イベント・トリガ・フレーム ID は 0x0～0x3B（パリティ含まない）でなければなりません。
3. 最初のデータバイトは関連する無条件フレームの PID が格納される為、データ領域としては、先頭 1 バイトを除いた最大 7 バイト分の領域を割りあてることが可能です。先頭 1 バイトには信号を割り当てないようにしてください。
4. イベント・トリガに関連付けられた無条件フレームのチェックサム・モデルは同じにしてください。（すなわち、LIN1.3 対応のスレーブ・ノードと LIN 2.1 対応のスレーブ・ノードを混在させることはできません。）
5. イベント・トリガに関連付けられた無条件フレームのデータ長は同じにしてください。
6. イベント・トリガに関連付けられた無条件フレームは、すべて異なるスレーブ・ノードから発行する必要があります。
7. LIN 2.1 マスター・ドライバでは、イベント・トリガ・フレームの衝突を検出した場合に衝突解決スケジュール・テーブルへ切り替えを行います。
8. LIN 2.1 マスター・ドライバでは、イベント・トリガ・フレームの衝突解決は無条件フレームを使用して行われます。
9. 衝突解決中の無条件フレームでエラーが発生してもこれを無視します。

7.3.3 スポラディック・フレーム転送

マスター・ノードは、スポラディック・フレームを用いて、関連付けられた無条件フレームのレスポンスを送受信します。

無条件フレームの送受信はシグナル更新があった場合のみ行います（マスターが無条件フレームのヘッダを送信します。スレーブは、ヘッダに応じた送受信を行います）。

複数の関連する無条件フレームにシグナル更新があった場合は、それらの中で最も優先度の高い無条件フレームを送受信します。

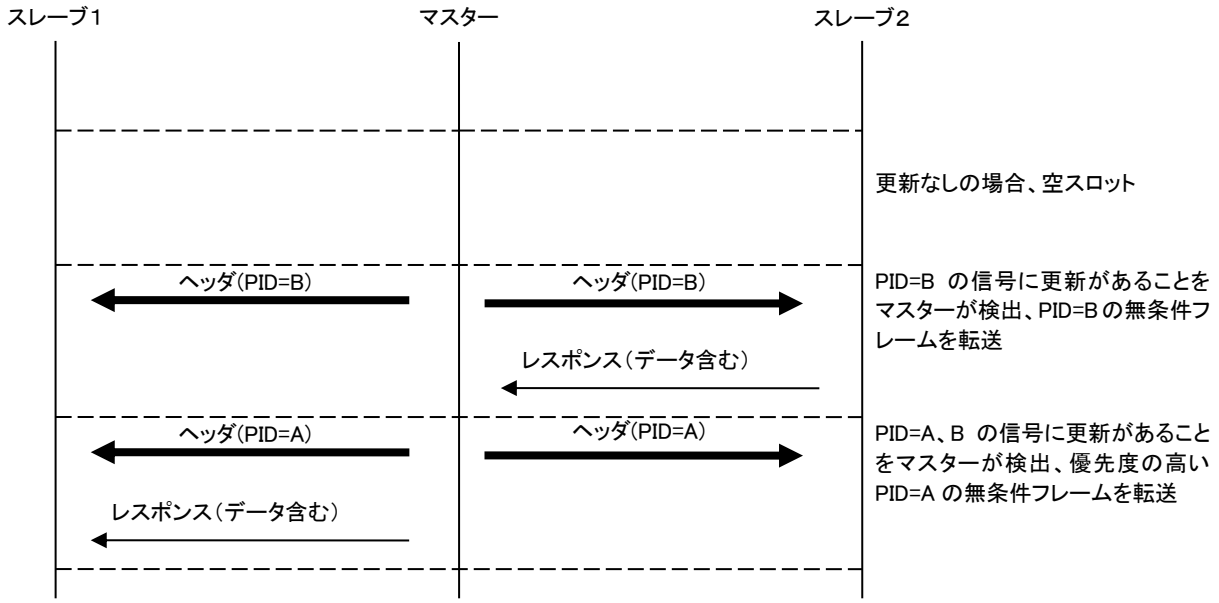
スポラディック・フレームのシグナル更新とは以下のいずれかの処理をさします。

・通信方向が送信の無条件フレームに対するシグナル書き込み

・イベント・トリガ・フレームに関連付けられた、通信方向が受信の無条件フレームをイベント・トリガ・フレームとして受信したいときに衝突を検出

スポラディック・フレームに関する無条件フレームを PID=A（送信）、B（受信）（A の優先度：高）とすると、以下のような通信が実施されます。

図7-7　スポラディック･フレームによるデータ送受信例



※注意：

1．関連する無条件フレームすべてにシグナル更新がなかった場合、ヘッダを送信しません。

2．スポラディック・フレームに関連付ける無条件フレームに制限はありません。チェックサム・モデル、データ長が異なったフレーム、割り振られたノードが同じフレームを、同じスポラディック・フレームに関連付けることができます。

7.4　レスポンス・エラー通知機能　【マスター/スレーブ】

レスポンス・エラー・シグナルとして特定の 1 ビット・シグナルを割り当てることにより、マスター・ノードからスレーブ・ノード内のエラー情報を読み出すことが可能です。

スレーブ・ノードにおいて、イベント･トリガ・フレームを除いたレスポンス・フィールドでエラーが検出されると、レスポンス･エラー･シグナルに“１”がセットされます。

マスター･ノードへ送信するまでは“1”が保持され、マスター･ノードへ送信された後“0”クリアされます。

マスター・ノードは以下の情報でスレーブ・ノードのエラー情報を確認することができます。

表 7-3　レスポンス・エラー一覧

マスター受信値	意味
Response_Error = 0(False)	正常動作
Response_Error = 1(True)	一時的なエラーあり
受信なし	致命的エラーあり

※注意：

1. レスポンス・エラー・シグナルは、必ずサイズ 1 ビットである必要があります。1 ビットでない場合、正しい情報を読み出すことができません。
2. スレーブ・アプリケーションの場合、レスポンス・エラー・シグナルを必ず割り当ててください。
3. レスポンス・エラー・シグナルは、イベント・トリガ・フレームに配置しないでください。
4. レスポンス・エラー・シグナルの初期値は、コンフィギュレーション時に“0”で定義してください。
5. レスポンス・エラー・シグナルは、ユーザ･アプリケーション内で操作しないでください。

7.5　スリープ・ウェイクアップ機能　【マスター/スレーブ】

7.5.1　スリープ機能

LIN2.1スレーブ・ドライバがgo-to-sleep-commandを受信した場合、または4秒※1以上バスオフ状態が続いた場合、スレーブ・ノードはスリープ・モードに移行します。

スリープ・モード移行後はウェイクアップLowパルス取得待ち状態となります。

LIN2.1マスター・ドライバでgo-to-sleep-commandを正常に発行すると、スリープ・モードへ移行します。スリープ・モード移行後は、ウェイクアップ待ち状態となり、フレームの転送を行いません。

go-to-sleep-commandはノード・コンフィギュレーション機能のマスター・リクエスト・フレームとして送信するため、スケジュール・スロットにマスター・リクエスト・フレームを割り当て、l_ifc_goto_sleep関数をコールすると送信することができます。

RL78/F23,F24用ドライバでは、ウェイクアップ方法として「ドミナント幅検出」を選択した場合、ドライバ・スリープ中にもLINマクロ(RLIN3)へのクロック供給が必要です。

※1：時間はコンフィグ・ファイルにて設定してください。バスオフ検出なしに設定することも可能です。
また、バスオフを判定する処理にタイマ割り込みを利用しているため、ユーザ・アプリケーションで使用する割り込みとの競合などにより、スリープ・モードへの移行時間が伸びる可能性があります。

7.5.2　ウェイクアップ機能

スリープ・モード中に立ち下がりエッジを検出した場合、または l_ifc_wake_up 関数を発行した場合に、ウェイクアップを行います。

・立ち下がりエッジ検出の場合

スレーブ・ノードは、Break&Synch 待ちとなります。

マスター・ノードは、l_sch_tick 関数によりスケジュールの再開が可能となります。

・l_ifc_wake_up 関数を発行した場合

LIN バスに 260μs のウェイクアップ・パルスを送信します。

※19200 bps で 5 Tbit。

ウェイクアップ・パルス送信後、150 ms 以内に Break を受信しなかった場合、再度ウェイクアップを送信します。3 回送信後、4 回目は 1.5 sec 後にウェイクアップを送信します。3 回の送信を 1 ブロックとし、ユーザ定義する回数分、ブロック送信を繰り返します。その後 4 sec 以上バス非アクティブ状態ならばスリープ・モードに移行します。

マスター・ノードは、ｌ_sch_tick 関数によりスケジュールの再開が可能となります。

※注意：

1．ウェイクアップからフレーム転送再開までのウェイト処理は、LIN2.1 ソフトウェア・ドライバで行いませ

ん。立ち下がりエッジ検出、または l_ifc_wake_up 発行後、アプリケーション・レベルで任意の時間ウェイトし、スケジュールを再開してください。

2. スリープ・モードからウェイクアップを行うと、スリープ・モードへ遷移する直前の状態となります。そのため l_sch_tick 関数をコールしたときに、必ずフレームが転送されるとは限りません。ウェイクアップ後は、LIN2.1 ソフトウェア・ドライバを初期化することを推奨します。

7.6 ノード・コンフィギュレーション機能 【マスター/スレーブ】

LIN2.1 ソフトウェア・ドライバのノード・コンフィギュレーション機能を用いて LIN クラスタに接続するスレーブ・ノードの設定が可能です。

スケジュール・スロットにマスター・リクエスト・フレームおよびスレーブ・レスポンス・フレームを割り当て、ld_assign_frame_id_range 関数、ld_read_by_id 関数をコールすると、スレーブ・ノードへコンフィギュレーション・コマンドを送信することができます。

7.6.1 ノード情報

スレーブ・ノードには、NAD、プロダクトID、レスポンス・エラー・シグナルのノード情報を定義する必要があります。コンフィグ・ファイルにて定義します。

表 7-4 ノード情報一覧

ノード情報	意味	サイズ
初期NAD	LINクラスタ内でのノード固有番号	１バイト
Supplier ID	LIN Consortiumから割り当てられるID	２バイト
Function ID	機能毎に割り当てられるID	２バイト
Variant ID	機能変更なしでの、製品バージョン毎に割り当てるID	１バイト
レスポンス・エラー・シグナル	スレーブ・ノード内のエラー情報を扱うシグナル	１バイト

7.6.2 ノード・コンフィギュレーション

LIN2.1ソフトウェア・ドライバでは、マスター・ノードによるノード・コンフィギュレーション機能の一部に対応しています。LIN2.1マスター・ドライバから、以下のリクエストを送信することが可能です。

・LIN フレーム ID 割り当て（Assign frame identifier range）
・識別子読み出し（Read by identifier）

（1） LIN フレーム ID 割り当て（Assign frame identifier range）

NAD が一致するノードの最大 4 つのメッセージフレームの PID を変更または無効にします。
ただし、ID が 60~63(0x3C～0x3F)のフレームは変更できません。
開始インデックスは PID を割り当てる最初のフレームを指定します。フレームの順序はスレーブ・ノードに依存します。リストの最初のインデックスは 0 から開始します。
PID リストの設定値は以下のとおりです。
マスター・リクエスト・フレームの通信フォーマットは以下の通りです。

表 7-5 マスター・リクエスト・フレーム・フォーマット (Assign frame identifier range)

NAD	PCI	SID	D1	D2	D3	D4	D5
NAD	0x06	0xb7	Start Index	PID (Index+0)	PID (Index+1)	PID (Index+2)	PID (Index+3)

スレーブからのレスポンスは NAD が一致した場合のみ送信されます。
全ての割り当てが実行されなかった場合、否定応答が返ります。
また、スレーブ・ノードでは指定された PID の検証を行わないため、マスター・ノードは正しい PID を設定する必要があります。

※注意：
正常終了時のスレーブ・レスポンス・フレーム転送には対応していません。
Protected ID はパリティ付きの ID を指定する必要があります。

（2） 識別子読み出し（Read by identifier）

マスター・ノードより ld_read_by_id 関数を発行することにより、以下のフォーマットでリクエストを発行します。その後スレーブ・レスポンス・フレームにより ID（D1：identifier）に従った情報をスレーブ・ノードから取得することが可能です。
マスター・リクエスト・フレームの通信フォーマットは以下の通りです。

表 7-6　マスター・リクエスト・フレーム・フォーマット (Read by identifier)

NAD	PCI	SID	D1	D2	D3	D4	D5
NAD	0x06	0xb2	Identifier	Supplier ID LSB	Supplier ID MSB	Function ID LSB	Function ID MSB

スレーブ・ノード内で NAD、Supplier ID と FunctionID が一致した場合のみ、スレーブはレスポンスを返します。ただし、NAD にワイルドカード（0x7F）が指定された場合は、Supplier ID と FunctionID が一致すれば、処理を実施します。

また、一致しない場合は、マスター・ノードからのスレーブ・レスポンス・リクエストに対応しません。

スレーブ・レスポンス・フレームの通信フォーマットは以下の通りです。

表 7-7　スレーブ・レスポンス・フレーム・フォーマット

ID
0

NAD	PCI	RSID	D1	D2	D3	D4	D5
NAD	0x06	0xf2	Supplier ID LSB	Supplier ID MSB	Function ID LSB	Function ID MSB	Variant

スレーブがマスターからのリクエストをサポートしていない場合は、以下の否定応答を返します。

表 7-8　否定応答フォーマット

ID
1 - 255

NAD	PCI	RSID	D1	D2	D3	D4	D5
NAD	0x03	0x7f	Requested SID(=0xb2)	Error code (= 0x12)	0xff	0xff	0xff

※　LIN2.1 スレーブ・ドライバでは、Identifier=0（プロダクト ID 読み出し）のみ対応しています。その他の Identifier には否定応答を返します。

7.7 スケジューリング機能 【マスター】

スケジューリング機能として用意された l_sch_tick 関数、l_sch_set 関数をコールすることで、コンフィグ・ファイルで定義したスケジュール・テーブルの遷移、切り替えが可能です。

スケジューラは、タイマと上記関数を用いてマスターのユーザ･アプリケーション内で実装する必要があります。「5. 2. 6 スケジューラの実装（マスターのみ）」を参照してください。

- スレーブ診断のタイミングパラメータについて

LIN 2.1 Spec では、スレーブ診断のタイミング要件として以下のパラメータが規定されています。

P2 : スレーブ・ノードが最後の診断要求を受信してから、応答用のデータを提供できる様になるまでの時間

STmin : スレーブ・ノードが診断要求のその次の診断要求の受信、または診断応答の次の診断応答の送信を準備するために必要な最小時間

P2* : スレーブ・ノードがネガティブ・レスポンスを送信してから、次の応答用のデータを提供できるようになるまでの時間

LIN マスター/スレーブ・ドライバでは上記パラメータに関して実装しておりません。
上記時間はマスター・ドライバのスケジュールで調整することになります。

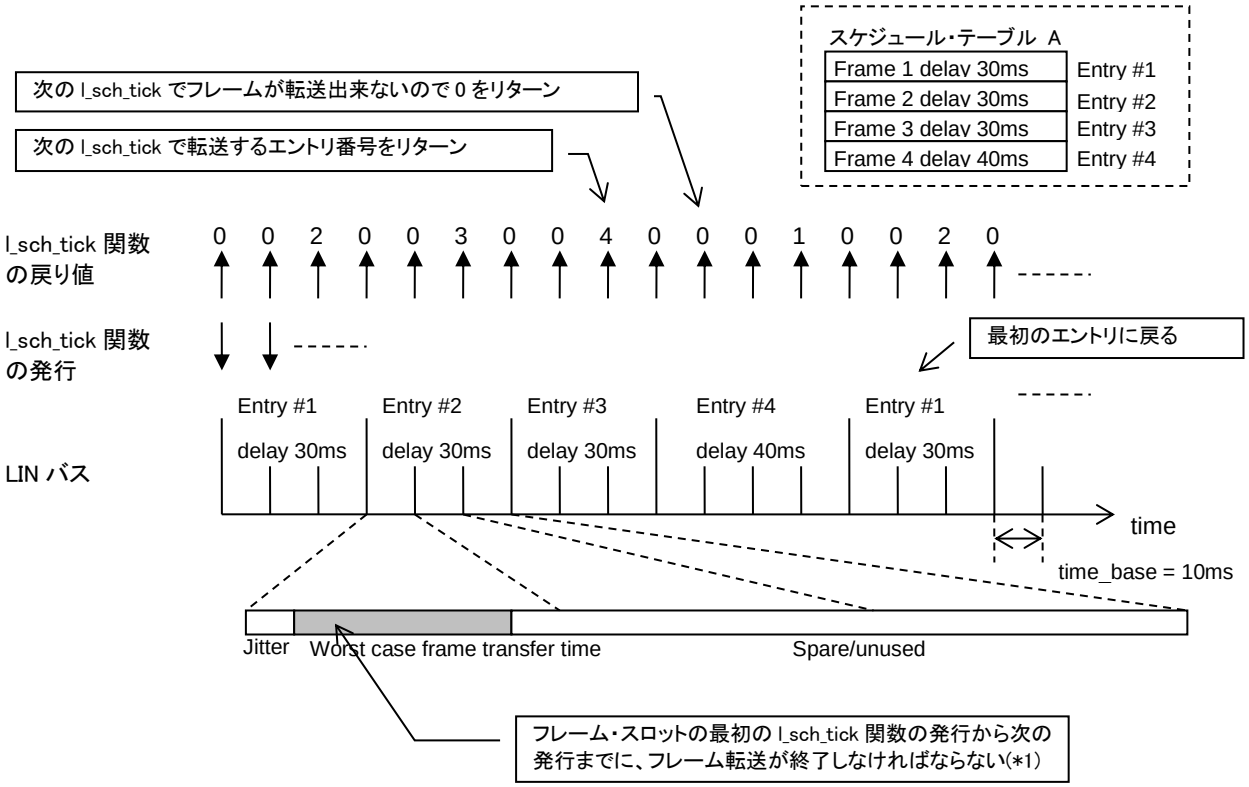
7.7.1 スケジュール遷移（l_sch_tick）

l_sch_tick関数を発行することにより、スケジュールが遷移します。l_sch_tick関数をコールし、スケジュール・テーブルにエントリされている現在のフレーム・スロットが終了すると、次のエントリのフレーム転送を開始します（終了したフレーム・スロットがスケジュール・テーブルの最後のエントリのときは、スケジュール・テーブルの最初のエントリのフレーム転送を開始します）。フレーム・スロットが終了しないときは、フレーム転送を開始しません。

次のl_sch_tick関数の発行で、次エントリのフレーム転送を開始するとき（現在のフレーム・スロットが終了するとき）、そのエントリ番号を返します。次のl_sch_tick関数のコールでフレーム転送が開始できないときは、0を返します。（図5 - 9参照）

4つのエントリで構成されるスケジュール・テーブルAがセットされているとき、l_sch_tick関数の発行とフレーム転送の関係を以下に示します。

図7-8　l_sch_tick関数とフレーム転送の関係



(*1):フレーム転送中の l_sch_tick コールは許可されていません。

補足説明:スケジュール･テーブル A には 4 つのフレーム･スロットがエントリされています。
time_base はフレーム転送の最大許容時間です。
time_base が必ずフレーム転送時間よりも長い時間となるように、スケジューラを設定してください。

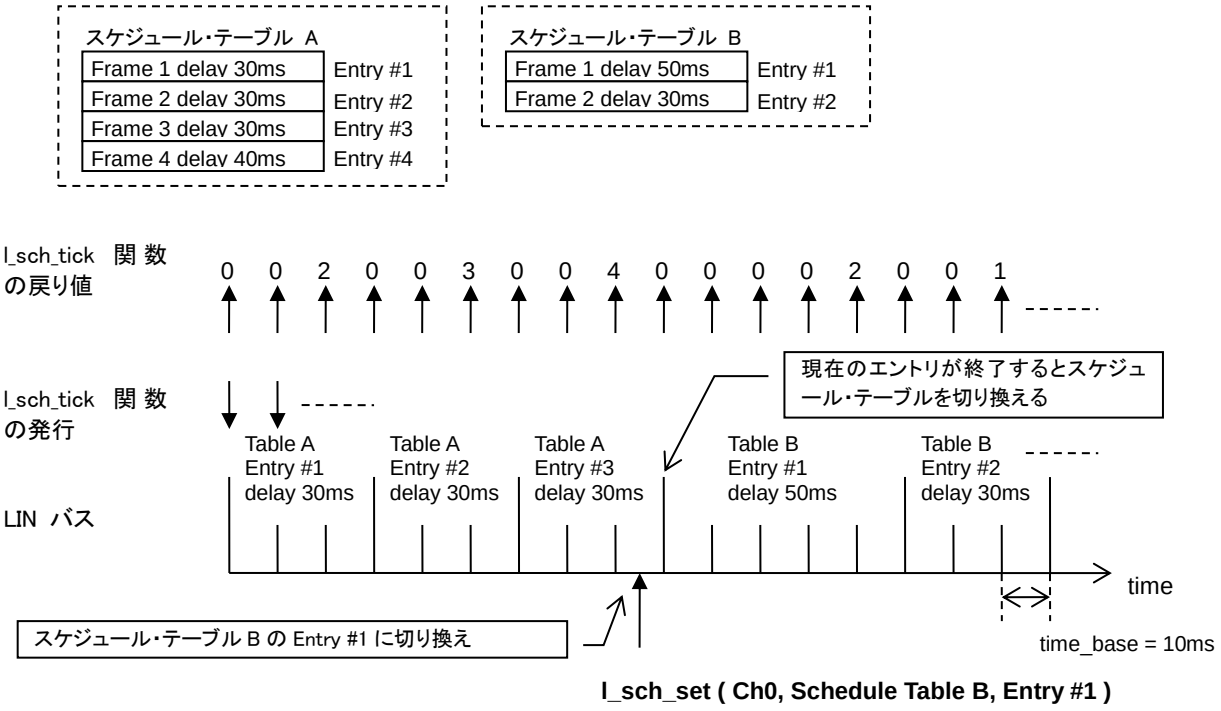
7.7.2 スケジュール切り替え（l_sch_set）

l_sch_set関数を発行すると、動作中のスケジュール・テーブルから指定したスケジュール・テーブルに切り替えることができます。また、切り替えるスケジュール・テーブルのエントリも指定できます。

スケジュール・テーブルの切り替えは、現在のスケジュール・テーブル・エントリのフレーム・スロットが終了したときに行います。

l_sch_set関数を発行して、スケジュール・テーブルAからスケジュール・テーブルBに切り替える処理を以下に示します。

図7-9 l_sch_set関数によるスケジュール・テーブルの切り替え



7.8 ボー・レート自動検出機能【スレーブ】

LIN2.1 スレーブ・ドライバでは、Break と Synch 幅を計測することにより、2400bps～20000bps までの通信ボー・レートの自動認識を行えるようコンフィギュレーションが可能です。

7.9 ドライバ・コンフィグレーション

7.9.1 スレーブ・ドライバ・コンフィグレーション

RL78/F23,F24のスレーブ・ドライバでは、ドライバ・コンフィグレーションによって、ドライバの動作を変更することが可能です。ドライバ・コンフィグレーションはlibsrc/conf/confslin_opt.h内の“ドライバ・コンフィグレーション”記載がある箇所で設定します。

表 7-9 ドライバ・コンフィグレーション(RL78/F23,F24 スレーブ) (1/2)

コンフィグ項目 (接頭子”CONFSLIN_OPT”を省略)	概要	設定可能値(接頭子”CONFSLIN_OPT”を省略)	設定可能値の意味
u1gLPRS_NORM_CFG	LIN マクロに供給するプリスケーラ･クロックの分周値です。**※1、※4**	u1gLPRS_NODIV	1/1
		u1gLPRS_DIV2P1	1/2
		u1gLPRS_DIV2P2	1/4
		u1gLPRS_DIV2P3	1/8
		u1gLPRS_DIV2P4	1/16
		u1gLPRS_DIV2P5	1/32
		u1gLPRS_DIV2P6	1/64
		u1gLPRS_DIV2P7	1/128
u1gBUSWKUP_CFG ※6	LIN ドライバのウェイクアップ方法です。EDGE を選択した場合、スリープ中の LIN マクロへのクロック供給は不要です。	u1gBUSWKUP_EDGE	LIN バスの立ち下がりエッジによるウェイクアップ (ドライバ・スリープ中、LIN マクロにクロックは供給されません。)
		u1gBUSWKUP_WIDTH	LIN バスのドミナント幅検出によるウェイクアップ
u1gNSPB_NORM_CFG	ビット･サンプリング数です。**※4**	u1gNSPB_4SMPL	4(オート・ボー・レート専用)
		u1gNSPB_8SMPL	8(オート・ボー・レート専用)
		u1gNSPB_16SMPL	16(固定ボー・レート専用)
u1gLINMCK_CFG	LIN 通信クロック源の種別です。**※5**	u1gLINMCK_FCLK	f_{CLK}
		u1gLINMCK_FMX	f_{MX} **※2**
u1gINTLINTRMPR_CFG ※6	LIN 送信完了割り込みの優先度です。	u1gINTPR_LV0	レベル 0 (最高)
		u1gINTPR_LV1	レベル 1
		u1gINTPR_LV2	レベル 2
		u1gINTPR_LV3	レベル 3 (最低)
u1gINTLINRVCPR_CFG ※6	LIN 受信完了割り込みの優先度です。	u1gINTLINTRMPR_CFG と同じです。	-
u1gINTLINSTAPR_CFG ※6	LIN ステータス割り込みの優先度です。	u1gINTLINTRMPR_CFG と同じです。	-
u1gTMUNIT_CFG ※6	使用するインターバル･タイマ(TAU)のユニット番号です。**※3**	u1gTMUNIT_UNIT0	ユニット 0
		u1gTMUNIT_UNIT1	ユニット 1
u1gTMCH_CFG ※6	使用するインターバル･タイマ(TAU)のチャネル番号です。**※3**	u1gTMCH_CH0	チャネル 0
		u1gTMCH_CH1	チャネル 1
		u1gTMCH_CH2	チャネル 2
		u1gTMCH_CH3	チャネル 3
		u1gTMCH_CH4	チャネル 4
		u1gTMCH_CH5	チャネル 5
		u1gTMCH_CH6	チャネル 6
		u1gTMCH_CH7	チャネル 7

u1gINTTMPR_CFG　※6	インターバル・タイマ(TAU)の割り込み優先度です。	u1gINTLINTRMPR_CFG と同じです。	-
u1gINTPPR_CFG　※6	外部割り込みの優先度です。	u1gINTLINTRMPR_CFG と同じです。	-

※1 : オート・ボー・レート・モードの場合、この設定値は使用されません。プリスケーラ・クロックが 8～12[MHz]になるように、ソフトウェアで自動設定されます。

※2 :　LIN コンフィグレータの周辺ハードウェア・クロック設定値は f_{MX} の値となります。この場合、インターバル・タイマ(TAU)の使用クロックは f_{CLK} で固定のため、LIN コンフィグレータでコード生成後、下記設定を修正する必要があります。

(conflin_x.c)

【修正前】

```
const u2 ConfSLin_u2gTMPERICLOCK       = (u2)CONFLIN_u2sPERICLOCK;
```

【修正後】　※f_{CLK} として 8[MHz]を使用する場合。

```
const u2 ConfSLin_u2gTMPERICLOCK       = (u2)800;
```

※3 :　使用するマイクロコンピュータのデバイスに搭載されているユニット、チャネルを設定してください。ドライバ内では使用可否のチェックは行っていません。

※4 :　ドライバではボー・レート・プリスケーラレジスタ(LBRP)の設定値を求めるために、内部で以下の計算を行います。

(LBRP 設定値) = {(LIN 通信クロック源周波数[Hz]) x (LPRS で設定するプリスケーラ分周)} /
{(ビット・サンプリング数) x (通信ボー・レート[bps])}

上記計算は整数(小数点以下切り捨て)で行われます。(LBRP 設定値)が 0 以下にならないように各パラメータを設定してください。

※5 :　LIN/UART モジュール(RLIN3)において、LIN 通信クロック源に関する注意事項があります。詳細につきましてはユーザーズ・マニュアルを参照してください。本件に伴い、スレーブ・ドライバに以下の制限事項があります。

LIN 通信クロック源の種別（u1gINTLINTRMPR_CFG）に f_{MX}（u1gLINMCK_FMX）を設定する場合、タイムアウト・エラー検出の使用選択（u1gTER_CFG）を検出しない（u1gTER_DISABLE）に設定した上で、f_{CLK} クロック ≧ f_{MX} クロック × 1.2 となるように設定してください。

※6 : スマート・コンフィグレータ上で設定することが可能です。

表 7-10　ドライバ・コンフィグレーション(RL78/F23,F24 スレーブ) (2/2)

コンフィグ項目 (接頭子"CONFSLIN_OPT"を省略)	概要	設定可能値(接頭子"CONFSLIN_OPT"を省略)	設定可能値の意味
u1gLRDNFS_NORM_CFG	LIN 通信時の受信ノイズフィルタ使用選択です。	u1gLRDNFS_USE	使用する
		u1gLRDNFS_NOUSE	使用しない
u1gLRDNFS_WKUP_CFG	ドライバ・スリープ中のウェイクアップ・パルス時の受信ノイズフィルタの使用選択です。	u1gLRDNFS_USE	使用する
		u1gLRDNFS_NOUSE	使用しない
u1gBLT_CFG	ブレーク・フィールドの受信を検出する最小のドミナント幅です。	u1gBLT_SHORT	[オート・ボー・レート・モード] 10[Tbit] [固定ボー・レート・モード] 9.5[Tbit]
		u1gBLT_LONG	[オート・ボー・レート・モード] 11[Tbit] [固定ボー・レート・モード] 10.5[Tbit]
u1gRS_CFG	送信時のレスポンス・スペース幅です。	u1gRS_0BIT	0[Tbit]
		u1gRS_1BIT	1[Tbit]
		u1gRS_2BIT	2[Tbit]
		u1gRS_3BIT	3[Tbit]
		u1gRS_4BIT	4[Tbit]
		u1gRS_5BIT	5[Tbit]
		u1gRS_6BIT	6[Tbit]
		u1gRS_7BIT	7[Tbit]
u1gIBS_CFG	送信時のインター・バイト・スペース幅です。	u1gIBS_0BIT	0[Tbit]
		u1gIBS_1BIT	1[Tbit]
		u1gIBS_2BIT	2[Tbit]
		u1gIBS_3BIT	3[Tbit]
u1gWUTL_CFG	送信ウェイクアップ・パルス幅です。	u1gWUTL_1BIT	1[Tbit]
		u1gWUTL_2BIT	2[Tbit]
		u1gWUTL_3BIT	3[Tbit]
		u1gWUTL_4BIT	4[Tbit]
		u1gWUTL_5BIT	5[Tbit]
		u1gWUTL_6BIT	6[Tbit]
		u1gWUTL_7BIT	7[Tbit]
		u1gWUTL_8BIT	8[Tbit]
		u1gWUTL_9BIT	9[Tbit]
		u1gWUTL_10BIT	10[Tbit]
		u1gWUTL_11BIT	11[Tbit]
		u1gWUTL_12BIT	12[Tbit]
		u1gWUTL_13BIT	13[Tbit]
		u1gWUTL_14BIT	14[Tbit]
		u1gWUTL_15BIT	15[Tbit]
		u1gWUTL_16BIT	16[Tbit]
u1gBERE_CFG	ビット・エラー検出の使用選択です。	u1gBERE_DISABLE	検出しない
		u1gBERE_ENABLE	検出する
u1gTER_CFG	タイムアウト・エラー検出の使用選択です。 **※1**	u1gTER_DISABLE	検出しない
		u1gTER_ENABLE	検出する (オート・ボー・レート・モード時は選択禁止です)
u1gFERE_CFG	フレーミング・エラー検出の使用選択です。	u1gFERE_DISABLE	検出しない
		u1gFERE_ENABLE	検出する
u1gSFERE_CFG	シンク・フィールド・エラー検出の使用選択で	u1gSFERE_DISABLE	検出しない
		u1gSFERE_ENABLE	検出する

	す。		
u1gIPERE_CFG	ID パリティ・エラー検出の使用選択です。	u1gIPERE_DISABLE	検出しない
		u1gIPERE_ENABLE	検出する
u1gLTES_CFG	タイムアウト・エラー検出の種別を選択します。 タイムアウト・エラー検出を使用しない場合、無関係です。	u1gLTES_FRAMETO	フレーム・タイムアウト
		u1gLTES_RESPTO	レスポンス・タイムアウト
u4gTRWTCNTMAX_CFG	LIN マクロ(RLIN3)の状態遷移を待つ時間(ソフトウェアカウンタ最大値)です。この設定値を超えても状態遷移しない場合、ドライバは通信を停止します。	1 以上	-
u2gTMCLKSEL_CFG	インターバル・タイマのクロック選択を指定します。	u2gTMCLKSEL_SEL0	クロック選択 0
		u2gTMCLKSEL_SEL1	クロック選択 1
u2gTMCLKDIV_CFG	インターバル・タイマの分周値を設定します。	u2gTMCLKDIV_NODIV	1/1
		u2gTMCLKDIV_DIV2	1/2
		u2gTMCLKDIV_DIV2P2	1/4
		u2gTMCLKDIV_DIV2P3	1/8
		u2gTMCLKDIV_DIV2P4	1/16
		u2gTMCLKDIV_DIV2P5	1/32
		u2gTMCLKDIV_DIV2P6	1/64
		u2gTMCLKDIV_DIV2P7	1/128
		u2gTMCLKDIV_DIV2P8	1/256
		u2gTMCLKDIV_DIV2P9	1/512
		u2gTMCLKDIV_DIV2P10	1/1024
		u2gTMCLKDIV_DIV2P11	1/2048
		u2gTMCLKDIV_DIV2P12	1/4096
		u2gTMCLKDIV_DIV2P13	1/8192
		u2gTMCLKDIV_DIV2P14	1/16384
		u2gTMCLKDIV_DIV2P15	1/32768

※1：**表 7-9　ドライバ・コンフィグレーション(RL78/F23,F24 スレーブ) (1/2)**の**※5** を参照してください。

7.9.2 マスター・ドライバ・コンフィグレーション

RL78/F23,F24のマスター・ドライバでは、ドライバ・コンフィグレーションによって、ドライバの動作を変更することが可能です。ドライバ・コンフィグレーションはlibsrc/conf/confmlin_opt.h内の“ドライバ・コンフィグレーション”記載がある箇所で設定します。

表 7-11 ドライバ・コンフィグレーション(RL78/F23,F24 マスター) (1/2)

コンフィグ項目 (接頭子”CONFMLIN_OPT”を省略)	概要	設定可能値(接頭子”CONFMLIN_OPT”を省略)	設定可能値の意味
u1gBUSWKUP_CFG ※3	LIN ドライバのウェイクアップ方法です。 EDGE を選択した場合、スリープ中の LIN マクロへのクロック供給は不要です。	u1gBUSWKUP_EDGE	LIN バスの立ち下がりエッジによるウェイクアップ(ドライバ･スリープ中､LIN マクロにクロックは供給されません。)
		u1gBUSWKUP_WIDTH	LIN バスのドミナント幅検出によるウェイクアップ
u1gLINMCK_CFG	LIN 通信クロック源の種別です。**※2**	u1gLINMCK_FCLK	f_{CLK}
		u1gLINMCK_FMX	f_{MX}
u1gBDT_CFG	送信するブレーク・デリミタの幅です。	u1gBDT_1BIT	1[Tbit]
		u1gBDT_2BIT	2[Tbit]
		u1gBDT_3BIT	3[Tbit]
		u1gBDT_4BIT	4[Tbit]
u1gIBHS_CFG	送信するヘッダ内のシンク・フィールドと ID フィールド間の幅と、レスポンス送信時のレスポンス・スペースの幅です。**※1**	u1gIBHS_0BIT	0[Tbit]
		u1gIBHS_1BIT	1[Tbit]
		u1gIBHS_2BIT	2[Tbit]
		u1gIBHS_3BIT	3[Tbit]
		u1gIBHS_4BIT	4[Tbit]
		u1gIBHS_5BIT	5[Tbit]
		u1gIBHS_6BIT	6[Tbit]
		u1gIBHS_7BIT	7[Tbit]
u1gIBS_CFG	レスポンス送信時の各バイト間の幅です。**※1**	u1gIBS_0BIT	0[Tbit]
		u1gIBS_1BIT	1[Tbit]
		u1gIBS_2BIT	2[Tbit]
		u1gIBS_3BIT	3[Tbit]
u1gINTLINTRMPR_CFG ※3	LIN 送信完了割り込みの優先度です。 go-to-sleep コマンドの送信と､ウェイクアップ･リクエスト送信時にのみ使用します。	u1gINTPR_LV0	レベル 0 (最高)
		u1gINTPR_LV1	レベル 1
		u1gINTPR_LV2	レベル 2
		u1gINTPR_LV3	レベル 3 (最低)
u1gINTLINRVCPR_CFG ※3	LIN 受信完了割り込みの優先度です。 ウェイクアップ方法が LIN バスのドミナント幅検出で､かつ､ウェイクアップ・リクエスト受信時にのみ使用します。	u1gINTLINTMRPR_CFG と同じです。	-
u1gINTLINSTAPR_CFG ※3	LIN エラー割り込みの優先度です。 go-to-sleep コマンドの送信と､ウェイクアップ･リクエスト送信時にのみ使用します。	u1gINTLINTMRPR_CFG と同じです。	-
u1gINTPPR_CFG ※3	外部割り込みの優先度です。 ウェイクアップ方法が LIN バスの立ち下がりエッジで､かつ､ウェイクア	u1gINTLINTMRPR_CFG と同じです。	-

	ップ・リクエスト受信時にのみ使用します。		

※1：フレーム・タイムアウト・エラー検出を有効にしている場合、フレームの時間が以下を超えるとタイムアウト・エラーを検出します。超えないように値を設定してください。

・クラシック・チェックサム選択時　：49 + (レスポンス・バイト数 + 1) x 14 [ビット]
・エンハンスド・チェックサム選択時：48 + (レスポンス・バイト数 + 1) x 14 [ビット]

レスポンス・タイムアウト・エラー検出を有効にしている場合、レスポンスの時間が以下を超えるとタイムアウト・エラーを検出します。超えないように値を設定してください。

・(レスポンス・バイト数 + 1) x 14 [ビット]

※2：RL78/F23,F24 の LIN/UART モジュール(RLIN3)において、LIN 通信クロック源に関する注意事項があります。詳細につきましてはユーザーズ・マニュアルを参照してください。本件に伴い、マスター・ドライバに以下の制限事項があります。

LIN 通信クロック源の種別（u1gINTLINTRMPR_CFG）に f_{MX}（u1gLINMCK_FMX）を設定する場合、タイムアウト・エラー検出の使用選択（u1gTER_CFG）を検出しない（u1gTER_DISABLE）に設定した上で、f_{CLK} クロック ≧ f_{MX} クロック × 1.2 となるように設定してください。

※3：スマート・コンフィグレータ上で設定することが可能です。

表 7-12　ドライバ・コンフィグレーション(RL78/F23, F24 マスター) (2/2)

コンフィグ項目 (接頭子"CONFMLIN_OPT"を省略)	概要	設定可能値(接頭子"CONFMLIN_OPT"を省略)	設定可能値の意味
u1gLRDNFS_NORM_CFG	LIN 通信時の受信ノイズフィルタ使用選択です。	u1gLRDNFS_USE	使用する
		u1gLRDNFS_NOUSE	使用しない
u1gWUTL_CFG	送信ウェイクアップ・パルス幅です。※1	u1gWUTL_1BIT	1[Tbit]
		u1gWUTL_2BIT	2[Tbit]
		u1gWUTL_3BIT	3[Tbit]
		u1gWUTL_4BIT	4[Tbit]
		u1gWUTL_5BIT	5[Tbit]
		u1gWUTL_6BIT	6[Tbit]
		u1gWUTL_7BIT	7[Tbit]
		u1gWUTL_8BIT	8[Tbit]
		u1gWUTL_9BIT	9[Tbit]
		u1gWUTL_10BIT	10[Tbit]
		u1gWUTL_11BIT	11[Tbit]
		u1gWUTL_12BIT	12[Tbit]
		u1gWUTL_13BIT	13[Tbit]
		u1gWUTL_14BIT	14[Tbit]
		u1gWUTL_15BIT	15[Tbit]
		u1gWUTL_16BIT	16[Tbit]
u1gBERE_CFG	ビット・エラー検出の使用選択です。	u1gBERE_DISABLE	検出しない
		u1gBERE_ENABLE	検出する
u1gPBERE_CFG	フィジカル･バス･エラー検出の使用選択です。	u1gPBERE_DISABLE	検出しない
		u1gPBERE_ENABLE	検出する
u1gTER_CFG	タイムアウト・エラー検出の使用選択です。**※2**	u1gTER_DISABLE	検出しない
		u1gTER_ENABLE	検出する
u1gFERE_CFG	フレーミング・エラー検出の使用選択です。	u1gFERE_DISABLE	検出しない
		u1gFERE_ENABLE	検出する
u1gLTES_CFG	タイムアウト・エラー検出の種別を選択します。 タイムアウト・エラー検出を使用しない場合、無関係です。	u1gLTES_FRAMETO	フレーム・タイムアウト
		u1gLTES_RESPTO	レスポンス・タイムアウト
u4gTRWTCNTMAX_CFG	LIN マクロ(RLIN3)の状態遷移を待つ時間(ソフトウェアカウンタ最大値)です。この設定値を超えても状態遷移しない場合、ドライバはユーザにエラーを通知します。	1 以上	-
u1gRXD_PU_CFG	LIN 受信端子の内部プルアップの使用選択です。	u1gRXD_PU_DISABLE	使用しない
		u1gRXD_PU_ENABLE	使用する
u1gRXD_PITHL_CFG	LIN 受信端子の入力閾値選択です。	u1gRXD_PITHL_03VDD	0.3 EV_{DD}
		u1gRXD_PITHL_03VDD	0.5 EV_{DD}

※1：6 ビット以下にしないでください。送信ウェイクアップ・パルス幅が LIN 仕様を満たせなくなる可能性があります。

※2：**表 7-11　ドライバ・コンフィグレーション(RL78/F23,F24 マスター) (1/2)**の**※2** を参照してください。

第8章 LIN2.1 ソフトウェア・ドライバ関数（スレーブ用）

8. 1 LIN2.1 ソフトウェア･スレーブ・ドライバ関数一覧

LIN2.1 ソフトウェア・スレーブ・ドライバ関数一覧を次に示します。

表 8-1 LIN2.1 ソフトウェア・スレーブ・ドライバ関数一覧

分類	関数名	機能概要
LIN2.1ソフトウェア・ドライバとクラスタ管理	l_sys_init	システム初期化
スカラ・シグナル読み出し	l_bool_rd	1ビット・スカラ・シグナル読み出し処理
	l_u8_rd	8ビット・スカラ・シグナル読み出し処理
	l_u16_rd	16ビット・スカラ・シグナル読み出し処理
スカラ・シグナル書き込み	l_bool_wr	1ビット・スカラ・シグナル書き込み処理
	l_u8_wr	8ビット・スカラ・シグナル書き込み処理
	l_u16_wr	16ビット・スカラ・シグナル書き込み処理
バイト・アレイ読み出し	l_bytes_rd	バイト・アレイ読み出し処理
バイト・アレイ書き込み	l_bytes_wr	バイト・アレイ書き込み処理
通知	l_flg_tst	シグナル更新フラグ確認
	l_flg_clr	シグナル更新フラグクリア
インタフェース・マネージメント	l_ifc_init	インタフェース初期化
	l_ifc_wake_up	ウェイクアップ発行
	l_ifc_read_status	ステータス読み出し
ユーザ定義コール・アウト	l_sys_irq_disable	割り込み禁止設定
	l_sys_irq_restore	割り込み復帰設定
	l_sys_call_sleep	スリープ状態遷移
	l_sys_call_wake_up	ウェイクアップ開始

8.2 データ・タイプ（スレーブ用）

LIN2.1spec で定義されている型と LIN2.1 スレーブ・ドライバで使用する型は以下の定義になっています。

表 8-2 LIN2.1spec と LIN2.1 スレーブ・ドライバ型定義の対応一覧

LIN2.1 specで規定されている型	LIN2.1ソフトウェア・ドライバが使用する型
l_bool	unsigned char
l_u8	unsigned char
l_u16	unsigned short
l_signal_handle	unsigned char
l_flag_handle	unsigned char
l_irqmask	unsigned char
l_ifc_handle	unsigned char

8.3 LIN2.1 ソフトウェア・スレーブ・ドライバ関数の説明

これより、LIN 2.1 ソフトウェア・スレーブ・ドライバ関数について図 7-1 の形式に従って解説します。

図 8-1 LIN2.1 ソフトウェア・スレーブ・ドライバ関数の記述フォーマット

1 →

○○○○○○

2 → 【概要】

・・・・・・・・・・・・・・・・・・・・・・・・・

3 → 【C 言語記述形式】

・・・・・・・・・・・・・・・・・・・・・・・・・

4 → 【パラメータ】

I/O	パラメータ	説明

5 → 【説明】

・・・・・・・・・・・・・・・・・・・・・・・・・

6 → 【戻り値】

エラー・コード	値	意味

7 → 【注意】

・・・・・・・・・・・・・・・・・・・・・・・・・

8 → 【使用例】

・・・・・・・・・・・・・・・・・・・・・・・・・・・

・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・

・・・・・・・・・・・・・・・・・・・・・・・・・

１．名称

LIN2.1 ソフトウェア・ドライバ関数の名称を示しています。

２．【概要】

LIN2.1 ソフトウェア・ドライバ関数の機能概要を示しています。

３．【C 言語記述形式】

LIN2.1 ソフトウェア・ドライバ関数を C 言語で発行する際の記述形式を示しています。

４．【パラメータ】

LIN2.1 ソフトウェア・ドライバ関数のパラメータを以下の形式で示しています。

I/O	パラメータ	説明
A	B	C

A：パラメータの入出力区分

I　…　入力パラメータ

O　…　出力パラメータ

B：パラメータの型および名称

C：パラメータの説明

５．【説明】

LIN2.1 ソフトウェア・ドライバ関数の機能を説明しています。

６．【戻り値】

LIN2.1 ソフトウェア・ドライバ関数の戻り値を以下の形式で示しています。

エラー・コード	値	意味
A	B	C

A：戻り値がエラー・コードの場合の名称

B：戻り値のとりうる値

C：戻り値の説明

７．【注意】

LIN2.1 ソフトウェア・ドライバ関数に関する注意事項です。主に実装依存による注意点について説明しています。

８．【使用例】

LIN2.1 ソフトウェア・ドライバ関数ごとの使用例です。

8.3.1 [スレーブ] LIN2.1 ソフトウェア・ドライバとクラスタ管理（1 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 8-3 LIN2.1 ソフトウェア・ドライバとクラスタ管理関数一覧（スレーブ）

関数名	機能概要
l_sys_init	システム初期化

l_sys_init

【概要】

LIN2.1 ソフトウェア・ドライバを初期化します。

【C 言語記述形式】

```
l_bool l_sys_init(void)
```

【パラメータ】

なし

【説明】

この関数は、LIN2.1 ソフトウェア・ドライバ・システムを初期化します。すべての関数を発行する前に呼び出す必要があります。

【戻り値】

エラー・コード	値	意味
L_SUCCESS	0x00	初期化成功

※LIN 2.1 ソフトウェア・ドライバでは、L_SUCCESS のみ戻り値として返します。
呼び出し元では、安全のため戻り値のチェックを行うことをお勧めします。

【使用例】

```
/* スタートアップ終了後 */
 if ( l_sys_init() ){
        /* init error */
        ;
 }
 else{
      /* 以後他の LIN2.1 ソフトウェア・ドライバ関数をコール可能 */
 }
```

8.3.2 [スレーブ] スカラ・シグナル読み出し（3 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 8-4 スカラ・シグナル読み出し関数一覧（スレーブ）

関数名	機能概要
l_bool_rd	1ビット・スカラ・シグナル読み出し処理
l_u8_rd	8ビット・スカラ・シグナル読み出し処理
l_u16_rd	16ビット・スカラ・シグナル読み出し処理

l_bool_rd

【概要】

1 ビットのスカラ・シグナルを読み出します。

【C 言語記述形式】

l_bool l_bool_rd(l_signal_handle sss)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle sss	シグナル名（マクロ値）

【説明】

この関数は、指定されたシグナル名に関連付けられた、メッセージ・バッファ内のシグナル・データを読み出します。LIN 通信とは非同期に読み出すことが可能です。

【戻り値】

エラー・コード	値	意味
-	0x00,0x01	シグナル・データ

【注意】

・1 ビット以外のサイズで定義したシグナルを読み出さないでください。

・sss は定義したシグナル名以外を指定しないでください。

・送信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_SWITCH
   ・サイズ 1 ビット
*/

if(l_bool_rd( WINDOW_SWITCH ))
{
    “ウィンドウスイッチ ON の場合の処理”
}
else
{
    “ウィンドウスイッチ OFF の場合の処理”
}
```

l_u8_rd

【概要】

1-8 ビットのスカラ・シグナルを読み出します。

【C 言語記述形式】

l_u8 l_u8_rd(l_signal_handle sss)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle sss	シグナル名（マクロ値）

【説明】

この関数は、指定されたシグナル名に関連付けられた、メッセージ・バッファ内のシグナル・データを読み出します。LIN 通信とは非同期に読み出すことが可能です。

新規データの有無に関わらず現在のシグナル・データを読み出します。

【戻り値】

エラー・コード	値	意味
-	0-0xFF	シグナル・データ

【注意】

・8 ビットより大きいサイズで定義したシグナルを読み出さないでください。この場合、シグナルの 8 ビット分のみ読み出されます。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_STATUS
   ・サイズ 8 ビット
*/

#define ST_BUSY (0x00)
#define ST_IDLE   (0x01)
#define ST_NG                    (0x02)

switch_status = l_u8_rd(WINDOW_STATUS);

if( switch_status == ST_BUSY )
{
        “BUSY 状態の処理”
}
else if( switch_status == ST_NG )
{
        “NG 状態の処理”
}
```

l_u16_rd

【概要】

1-16 ビットのスカラ・シグナルを読み出します。

【C 言語記述形式】

l_u16 l_u16_rd(l_signal_handle sss)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle sss	シグナル名（マクロ値）

【説明】

この関数は、指定されたシグナル名に関連付けられた、メッセージ・バッファ内のシグナル・データを読み出します。LIN 通信とは非同期に読み出すことが可能です。

新規データの有無に関わらず現在のシグナル・データを読み出します。

【戻り値】

エラー・コード	値	意味
-	0-0xFFFF	シグナル・データ

【注意】

・16 ビットより大きいサイズで定義したシグナルを読み出さないでください。

・sss には定義したシグナル名以外指定しないでください。

・送信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_STATUS
   ・サイズ 16 ビット
*/
#define ST_BUSY       0x0000
#define ST_IDLE       0x0001

l_u16 switch_status;

switch_status = l_u16_rd(WINDOW_STATUS);
switch( switch_status )
{
    case ST_BUSY:
        /* BUSY 状態の処理を記述します。 */
        break;
    case ST_IDLE:
        /* IDLE 状態の処理を記述します。 */
        break;
    default:
        /* 異常時の処理を記述します。 */
        break;
}
```

8.3.3 [スレーブ] スカラ・シグナル書き込み（3 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 8-5 スカラ・シグナル書き込み関数一覧（スレーブ）

関数名	機能概要
l_bool_wr	1ビット・スカラ・シグナル書き込み処理
l_u8_wr	8ビット・スカラ・シグナル書き込み処理
l_u16_wr	16ビット・スカラ・シグナル書き込み処理

l_bool_wr

【概要】

1 ビットのスカラ・シグナルを書き込みます。

【C 言語記述形式】

void l_bool_wr(l_signal_handle sss, l_bool v)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle　sss	シグナル名（マクロ値）
I	l_bool　　v	書き込みデータ

【説明】

この関数は、指定されたシグナル名に関連付けられたメッセージ・バッファ内に、v で指定したデータを書き込みます。

書き込み後、該当するシグナルの更新フラグをセットします。

【戻り値】

なし

【注意】

・1 ビットより大きいサイズで定義したシグナルを書き込まないでください。

・sss には定義したシグナル以外指定しないでください。

・受信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW、WINDOW_SWITCH
   ・サイズ 1 ビット
*/
#define PUSH      (0x01)
#define OPEN      (0x00)
/* スイッチプッシュを検出 */
if( l_bool_rd(WINDOW_SWITCH) == PUSH )
{
        /* 1 ビット・シグナル OPEN を書き込み */
      l_bool_wr( WINDOW, OPEN );
}
```

l_u8_wr

【概要】

1-8 ビットのスカラ・シグナルを書き込みます。

【C 言語記述形式】

void l_u8_wr(l_signal_handle sss, l_u8 v)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle　sss	シグナル名（マクロ値）
I	l_u8　　v	書き込みデータ

【説明】

この関数は、指定されたシグナル名に関連付けられたメッセージ・バッファ内に、v で指定したデータを書き込みます。

書き込み後、該当するシグナルの更新フラグをセットします。

【戻り値】

なし

【注意】

・8 ビットより大きいサイズで定義したシグナルを書き込まないでください。

・sss には定義したシグナル以外指定しないでください。

・受信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW、WINDOW_SWITCH
   ・サイズ：8 ビット
*/
#define OPEN      (0x00)
#define CLOSE     (0x01)
#define SLEEP     (0x02)
#define UP        (0x00)
#define DOWN      (0x01)

switch(l_u8_rd(WINDOW_SWITCH))
{
    case UP:
        l_u8_wr( WINDOW, OPEN );
        break;

    case DOWN:
        l_u8_wr( WINDOW, CLOSE );
        break;

    default:
        l_u8_wr( WINDOW, SLEEP );
        break;
}
```

l_u16_wr

【概要】

1-16 ビット・スカラ・シグナルを書き込みます。

【C 言語記述形式】

```
void l_u16_wr(l_signal_handle sss, l_u16 v)
```

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle sss	シグナル名（マクロ値）
I	l_u16　　v	書き込みデータ

【説明】

この関数は、指定されたシグナル名に関連付けられたメッセージ・バッファ内に、v で指定されたデータを書き込みます。

書き込み後、該当するシグナルの更新フラグをセットします。

【戻り値】

なし

【注意】

・16 ビットより大きいサイズで定義したシグナルを書き込まないでください。この場合、シグナルの 16 ビット分のみ書き込まれます。

・sss には定義したシグナル以外指定しないでください。

・受信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：VOLUME
   ・サイズ 12 ビット
*/

l_u16 vol;

/* ボリュームレベルを取得 */
vol =GetVolLevel( );

/* シグナルに設定 */
l_u16_wr( VOLUME, vol );
```

8.3.4 [スレーブ] バイト・アレイ読み出し（1 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 8-6 バイト・アレイ読み出し関数一覧（スレーブ）

関数名	機能概要
l_bytes_rd	バイト・アレイ読み出し処理

l_bytes_rd

【概要】

バイト・アレイ・データを読出します。

【C 言語記述形式】

void l_bytes_rd(l_signal_handle sss, l_u8 start, l_u8 count, l_u8* const data)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle sss	シグナル名（マクロ値）
I	l_u8 start	読出し開始バイト番号
I	l_u8 count	読出しバイト数
O	l_u8* const data	読出し格納先

【説明】

この関数は、指定されたシグナル名に関連付けられたメッセージ・バッファ内の、start バイト目から count 数分のバイト値を読み出します。

例えば、バイト・アレイが 0 から 6 で番号付けされた 7 バイト長のメッセージだと仮定します。このバイト・アレイの 3～4 番目から(0、1、2 番目を飛ばして)読み出すには、start に 3 を指定し、count に 2 を指定します。この場合、3 番目の値が data[0]に書き込まれ、4 番目の値が data[1]へ書き込まれます。

【戻り値】

なし

【注意】

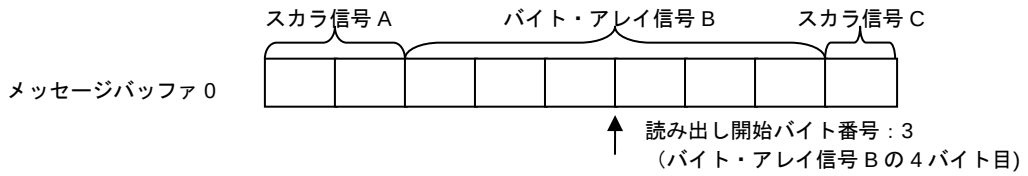
・start と count の組み合わせで、シグナル・サイズ範囲外となる指定をしないでください。正しい値を読み出すことができません。

・バイト・アレイとして定義したシグナルのみ読み出し可能です（オフセット及びサイズが 8 の倍数となるシグナル）。バイト・アレイでないシグナルを指定した場合、正しい値を読み出すことができません。

・送信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

・読み出し開始バイト番号は、メッセージ・バッファの先頭からの番号ではなく、シグナル名の先頭からの番号になります。下記の例を参照してください。

(例)バイト・アレイ信号 B の 4 バイト目から読み出したい場合



【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_TEST
   ・サイズ：4 バイト
*/

l_u8 data[4];

/* data 配列に 4 バイト分のデータが格納される */
l_bytes_rd(WINDOW_TEST, 0, 4, data)

/* data 配列に 2-3 バイト目までの 2 バイト分のデータが格納される */
l_bytes_rd(WINDOW_TEST, 2, 2, data);
```

8.3.5 [スレーブ] バイト・アレイ書き込み（1 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 8-7 バイト・アレイ書き込み関数一覧（スレーブ）

関数名	機能概要
l_bytes_wr	バイト・アレイ書き込み処理

l_bytes_wr

【概要】

バイト・アレイ・データを書き込みます。

【C 言語記述形式】

void l_bytes_wr(l_signal_handle sss, l_u8 start, l_u8 count, const l_u8* const data)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle　sss	シグナル名（マクロ値）
I	l_u8 start	書き込み開始バイト番号
I	l_u8 count	書き込みバイト数
I	const l_u8* const data	書き込みデータ格納先

【説明】

この LIN2.1 ソフトウェア・ドライバ関数は、指定されたシグナル名に関連付けられた、メッセージ・バッファ内の start バイト目から、count 数分のデータを書き込みます。

例えば、バイト・アレイが 0 から 6 で番号付けされた 7 バイト長のメッセージだと仮定します。このバイト・アレイのバイト 3 と 4 に書き込むには(バイト 0、1、2 を飛ばして)start を 3 にし、(バイト 3 と 4 を書き込む)count を 2 にする必要があります。この場合、バイト 3 は data[0]から読み出した値が書き込まれ、バイト 4 は data[1]から読み出した値が書き込まれます。

書き込み後、該当するシグナルの更新フラグをセットします。

【戻り値】

なし

【注意】

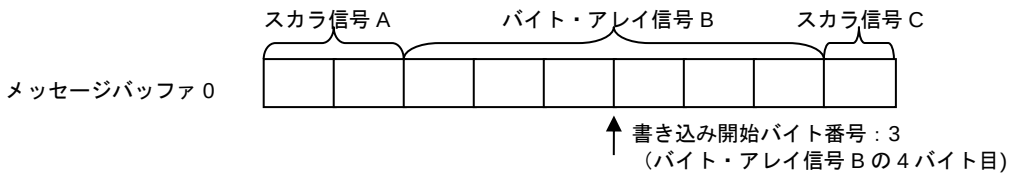
・start と count の組み合わせで、シグナル・サイズ範囲外となる指定をしないでください。正しい値を書き込むことができません。

・バイト・アレイとして定義したシグナルのみ書き込み可能です。（offset 及びサイズが 8 の倍数）その他の定義で指定した場合、正しい値を書き込むことができません。

・受信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

・書き込み開始バイト番号は、メッセージ・バッファの先頭からの番号ではなく、シグナル名の先頭からの番号になります。下記の例を参照してください。

(例)バイト・アレイ信号 B の 4 バイト目から書き込みたい場合



【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_TEST
   ・サイズ： 4 バイト
*/
const l_u8 data[4] = {0x00, 0x01, 0x02, 0x03};

/* data 配列の 4 バイト分のデータを書き込む */
l_bytes_wr(WINDOW_TEST, 0, 4, data);

/* WINDOW_TEST シグナルに 2 バイト分のデータを書き込む */
l_bytes_wr(WINDOW_TEST, 2, 2, data);
```

8.3.6 [スレーブ] 通知（2 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 8-8 通知関数一覧（スレーブ）

関数名	機能概要
l_flg_tst	シグナル更新フラグ確認
l_flg_clr	シグナル更新フラグクリア

l_flg_tst

【概要】

シグナル更新フラグの値を読み出します。

【C 言語記述形式】

l_bool l_flg_tst(l_flag_handle fff)

【パラメータ】

I/O	パラメータ	説　　明
I	l_flag_handle　fff	フラグ名（シグナル名と同等）

【説明】

この関数は、指定されたフラグ名に関連付けられたシグナル更新フラグを読み出します。

フラグが更新ありに設定されるタイミングは以下の通りです。

・正常受信完了した場合

・スカラ・シグナル、バイト・アレイ書き込み関数をコールし、シグナルにデータを書き込んだ場合

【戻り値】

エラー・コード	値	意味
-	0x00	更新なし
-	0x01	更新あり

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_STATUS
   ・サイズ： 1 バイト
*/

l_bool status;

if( l_flg_tst( WINDOW_STATUS ) )
{
        /* WINDOW_STATUS が更新された場合の処理を記載します */
        status = l_bool_rd( WINDOW_STATUS );
}
```

l_flg_clr

【概要】

シグナル更新フラグを 0 クリアします。

【C 言語記述形式】

void l_flg_clr(l_flag_handle fff)

【パラメータ】

I/O	パラメータ	説　　明
I	l_flag_handle　fff	フラグ名（マクロ値、シグナル名と同等）

【説明】

この LIN2.1 ソフトウェア・ドライバ関数は、指定されたフラグ名に関連付けられたシグナルの更新フラグをクリアします。

フラグがクリアされるタイミングは以下の通りです。

・本関数を発行した場合

・正常送信完了した場合

【戻り値】

なし

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_TEST
   ・サイズ：1 バイト
*/

l_u8 status;

if( l_flg_tst(WINDOW_TEST) )
{
    status = l_u8_rd(WINDOW_TEST);

      /* 更新フラグクリア */
    l_flg_clr(WINDOW_TEST);
}
```

8.3.7 [スレーブ] インタフェース・マネージメント（3 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 8-9 インタフェース・マネージメント関数一覧（スレーブ）

関数名	機能概要
l_ifc_init	インタフェース初期化
l_ifc_wake_up	ウェイクアップ発行
l_ifc_read_status	ステータス読み出し

l_ifc_init

【概要】

LIN インタフェースを初期化します。

【C 言語記述形式】

l_bool l_ifc_init (l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名（チャネル番号）

【説明】

この関数は、指定された LIN インタフェースに関連する初期化処理を実施し、指定したインタフェースの LIN 通信を使用可能にします。

LIN 送受信の許可処理を実施します。

【戻り値】

エラー・コード	値	意味
L_FAIL	0xFF	失敗
L_SUCCESS	0x00	成功

【注意】

・l_sys_init 関数発行後に使用してください。

・l_sys_init 関数以外の他の LIN2.1 ソフトウェア・ドライバ関数を使用する前に、必ず発行する必要があります。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください。

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
*/

if( l_sys_init( ) )
{
    /* エラー処理を記述します */
}
else
{
      /* インタフェース初期化 */
    if( l_ifc_init( LIN_CHANNEL0 ) )
    {
        /* エラー処理を記述します */
    }
}
```

l_ifc_wake_up

【概要】

ウェイクアップを発行します。

【C 言語記述形式】

void l_ifc_wake_up(l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名（チャネル番号）

【説明】

この関数は、指定された LIN インタフェースをスリープ・モードからウェイクアップに遷移し、LIN バスに 260μs のウェイクアップ Low パルスを送信します。

ウェイクアップ Low パルス送信後、150ms 以内に Break を受信しなかった場合、再度ウェイクアップを送信します。3 回ウェイクアップを繰り返した後、4 回目は 1.5sec 後にウェイクアップを送信します。3 回の送信を 1 ブロックとし、ユーザ定義する回数これを繰り返します。その後 4sec 以上バス非アクティブ状態ならばスリープ・モードに移行します。

【戻り値】

なし

【注意】

・スリープ・モードの場合のみ、発行することが可能です。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
   ･ステータス･マスク・ビット：LD_MASK_SLEEP
*/
/* スリープ・モードを検出したら、即ウェイクアップ */

/*スリープ検出 */
if( (l_ifc_read_status(LIN_CHANNEL0) & LD_MASK_SLEEP) == LD_MASK_SLEEP)
{
    l_ifc_wake_up();
}
```

l_ifc_read_status

【概要】

LIN2.1 ソフトウェア・ドライバの各種ステータスを読み出します。

【C 言語記述形式】

l_u16 l_ifc_read_status(l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名（チャネル番号）

【説明】

この関数は、指定された LIN インタフェースのステータス情報を読み出します。

【戻り値】

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit0
最終フレーム PID								0	Save configuration	Event triggered frame collision	Bus activity	Goto sleep	Overrun	Successful transfer	Error in response

Bit6-4 は LIN 2.1 でサポートされます。

最終フレーム PID　：最終通信フレームの PID を示します。

Save configuration　：対応していません。

Event triggered frame collision　：スレーブ・ドライバでは非該当ビットです。

Bus activity　：バスの立ち上がりエッジまたは立ち下りエッジを検出したときにセットされます。

Goto sleep　：go-to-sleep-command を正常送信した場合にセットされます。

Overrun　：l_ifc_read_status をコールする前に、２回以上の通信が発生した：場合にセットされます。

Successful transfer　：正常に通信が行われた場合セットされます。

Error in response　：レスポンス中に異常通信を検出した場合セットされます。

【注意】

・l_ifc_init 関数発行後に使用してください。

・本関数発行後または l_ifc_init 関数発行後、ステータスは 0 クリアされます。

・ステータスは蓄積フラグとなります。ひとつ前のこの関数コールから通信が複数回行われると、ステータスは上書きされます。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください。

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
   ･ステータス･マスク・ビット：LD_MASK_ERROR_IN_RESPONSE、LD_MASK_SUCCESSFUL_TRANSFER
*/

status = l_ifc_read_status( LIN_CHANNEL0 );
mask = LD_MASK_ERROR_IN_RESPONSE | LD_MASK_SUCCESSFUL_TRANSFER;

if( status& mask ) == mask )
{
    /* エラーと正常を両方検出、この場合断続的に通信が行われていると考えられる */
}
```

8.3.8 [スレーブ] ユーザ定義コール・アウト（4 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 8-10 ユーザ定義コール・アウト関数一覧（スレーブ）

関数名	機能概要
l_sys_irq_disable	割り込み禁止設定
l_sys_irq_restore	割り込み復帰設定
l_sys_call_sleep	スリープ状態遷移
l_sys_call_wake_up	ウェイクアップ開始

l_sys_irq_disable

【概要】

割込みを禁止します。

【C 言語記述形式】

l_irqmask l_sys_irq_disable(void)

【パラメータ】

なし

【説明】

この関数は割込みを禁止する為のコール・アウト関数です。

DI を実行し、現在の割り込み状態を返却します。返却値は、CC-RL 版のとき PSW の値、IAR 版のとき IAR 組み込み関数__get_interrupt_state によって返却された値です。

【戻り値】

I/O	値	意味
-	[CC-RL] PSWの割り込み状態ビット [IAR] グローバル割り込み状態	-

【注意】

本関数はコール・アウト関数なので変更可能ですが、スマート・コンフィグレータで提供するコードから変更しないでください。（「5. 2. 7 ユーザ定義コール・アウトの実装」参照）

【使用例】

自動的にコールされる為、アプリケーション内で使用する必要はありません。

l_sys_irq_restore

【概要】

割込み状態を元に戻します。

【C 言語記述形式】

void l_sys_irq_restore(l_irqmask previous)

【パラメータ】

I/O	パラメータ	説　　明
I	l_irqmask previous	割り込み復帰時の状態値 （[CC-RL] PSWの割り込み状態ビット [IAR] グローバル割り込み状態）

【説明】

この関数は、割込み状態を復帰させる為のコール・アウト関数です。

デフォルトでは、CC-RL 版のとき previous が割り込み許可設定（0x80）の場合 EI に設定し、それ以外は DI に設定します。IAR 版のときは IAR 組み込み関数__set_interrupt_state により、previous の指定状態に復帰します。

【戻り値】

なし

【注意】

本関数はコール・アウト関数なので変更可能ですが、スマート・コンフィグレータで提供するコードから変更しないでください。（「5. 2. 7 ユーザ定義コール・アウトの実装」参照）

【使用例】

自動的にコールされる為、アプリケーション内で使用する必要はありません。

l_sys_call_sleep

【概要】

スリープ・モード遷移時にコールされます。

本ソフトウェア独自の拡張関数です。

【C 言語記述形式】

void l_sys_call_sleep(l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名

【説明】

この関数は、スリープ・リクエストを受信した時、又はタイムアウトによりスリープ・モードに遷移した際にコール・アウトされる関数です。

【戻り値】

なし

【注意】

・スリープ・コマンドのデータ・フィールドの 1Byte 目を受信した直後にスリープ・モードに遷移するということはありません。必ずチェックサムまで受信した後にスリープ・モードに遷移します。

・本関数の実装は必須ではありません。使用しない場合は空処理としてください。スマート・コンフィグレータで提供されるコードでは、空処理の関数となっています。（「5. 2. 7 ユーザ定義コール・アウトの実装」参照）

【使用例】

自動的にコールされる為、アプリケーション内で使用する必要はありません。

l_sys_call_wake_up

【概要】

Wakeup 受信直後にコールされます。

本ソフトウェア独自の拡張関数です。

【C 言語記述形式】

void l_sys_call_wake_up(l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名

【説明】

この関数は、LIN2.1 ソフトウェア・ドライバがウェイクアップを受信した場合に、コール・アウトされる関数です。

【戻り値】

なし

【注意】

本関数の実装は必須ではありません。使用しない場合は空処理としてください。スマート・コンフィグレータで提供されるコードでは、空処理の関数となっています。（「5. 2. 7 ユーザ定義コール・アウトの実装」参照）

【使用例】

自動的にコールされる為、アプリケーション内で使用する必要はありません。

第9章 LIN2.1 ソフトウェア・ドライバ関数（マスター用）

9.1 LIN2.1 ソフトウェア・マスター・ドライバ関数一覧

LIN2.1 ソフトウェア・マスター・ドライバ関数一覧を次に示します。

表 9-1 LIN2.1 ソフトウェア・マスター・ドライバ関数一覧

分類	関数名	機能概要
LIN2.1ソフトウェア・ドライバとクラスタ管理	l_sys_init	システム初期化
スカラ・シグナル読み出し	l_bool_rd	1ビット・スカラ・シグナル読み出し処理
	l_u8_rd	8ビット・スカラ・シグナル読み出し処理
	l_u16_rd	16ビット・スカラ・シグナル読み出し処理
スカラ・シグナル書き込み	l_bool_wr	1ビット・スカラ・シグナル書き込み処理
	l_u8_wr	8ビット・スカラ・シグナル書き込み処理
	l_u16_wr	16ビット・スカラ・シグナル書き込み処理
バイト・アレイ読み出し	l_bytes_rd	バイト・アレイ読み出し処理
バイト・アレイ書き込み	l_bytes_wr	バイト・アレイ書き込み処理
通知	l_flg_tst	シグナル更新フラグ確認
	l_flg_clr	シグナル更新フラグクリア
スケジュール・マネージメント	l_sch_tick	スケジュール制御
	l_sch_set	スケジュール設定
インタフェース・マネージメント	l_ifc_init	インタフェース初期化
	l_ifc_goto_sleep	go-to-sleep-command発行
	l_ifc_wake_up	ウェイクアップ発行
	l_ifc_read_status	ステータス読み出し
ノード・コンフィギュレーション	ld_is_ready	ノード・コンフィギュレーション・レディ
	ld_check_response	ノード・コンフィギュレーション・チェック
	ld_assign_frame_id_range	PID割り当て
	ld_read_by_id	ID読み出し
ユーザ定義コール・アウト	l_sys_irq_disable	割り込み禁止設定
	l_sys_irq_restore	割り込み復帰設定
	l_sys_call_wake_up	ウェイクアップ開始
	l_sys_call_fatal_error	致命的エラー解決

9.2　データ・タイプ（マスター用）

LIN2.1spec で定義されている型と LIN2.1 マスター・ドライバで使用する型は以下の定義になっています。

表 9-2　LIN2.1spec と LIN2.1 マスター・ドライバ型定義の対応一覧

LIN2.1specで規定されている型	LIN2.1ソフトウェア・ドライバが使用する型
l_bool	unsigned char
l_u8	unsigned char
l_u16	unsigned short
l_signal_handle	unsigned char
l_flag_handle	unsigned char
l_irqmask	unsigned char
l_ifc_handle	unsigned char
l_ioctl_op	unsigned char
l_schedule_handle	unsigned char

9.3 LIN2.1 ソフトウェア・マスター・ドライバ関数の説明

これより、LIN2.1 ソフトウェア・マスター・ドライバ関数について図 8-1 の形式に従って解説します。

図9-1 LIN2.1ソフトウェア・マスター・ドライバ関数の記述フォーマット

1 →

○○○○○○

2 → 【概要】

・・・・・・・・・・・・・・・・・・・・・・・・・

3 → 【C 言語記述形式】

・・・・・・・・・・・・・・・・・・・・・・・・・

4 → 【パラメータ】

I/O	パラメータ	説明

5 → 【説明】

・・・・・・・・・・・・・・・・・・・・・・・・・

6 → 【戻り値】

エラー・コード	値	意味

7 → 【注意】

・・・・・・・・・・・・・・・・・・・・・・・・・

8 → 【使用例】

・・・・・・・・・・・・・・・・・・・・・・・・・・・・

・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・

・・・・・・・・・・・・・・・・・・・・・・・・・・・・

１．名称

LIN2.1 ソフトウェア・ドライバ関数の名称を示しています。

２．【概要】

LIN2.1 ソフトウェア・ドライバ関数の機能概要を示しています。

３．【C 言語記述形式】

LIN2.1 ソフトウェア・ドライバ関数を C 言語で発行する際の記述形式を示しています。

４．【パラメータ】

LIN2.1 ソフトウェア・ドライバ関数のパラメータを以下の形式で示しています。

I/O	パラメータ	説明
A	B	C

A：パラメータの入出力区分

I　…　入力パラメータ

O　…　出力パラメータ

B：パラメータの型および名称

C：パラメータの説明

５．【説明】

LIN2.1 ソフトウェア・ドライバ関数の機能を説明しています。

６．【戻り値】

LIN2.1 ソフトウェア・ドライバ関数の戻り値を以下の形式で示しています。

エラー・コード	値	意味
A	B	C

A：戻り値がエラー・コードの場合の名称

B：戻り値の取り得る値

C：戻り値の説明

７．【注意】

LIN2.1 ソフトウェア・ドライバ関数に関する注意事項です。主に実装依存による注意点について説明しています。

８．【使用例】

LIN2.1 ソフトウェア・ドライバ関数ごとの使用例です。

9.3.1 [マスター] LIN2.1 ソフトウェア・ドライバとクラスタ管理（1 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 9-3 LIN2.1 ソフトウェア・ドライバとクラスタ管理関数一覧（マスター）

関数名	機能概要
l_sys_init	システム初期化

l_sys_init

【概要】

LIN2.1 ソフトウェア・ドライバを初期化します。

【C 言語記述形式】

```
l_bool l_sys_init(void)
```

【パラメータ】

なし

【説明】

この関数は、LIN2.1 ソフトウェア・ドライバ・システムを初期化します。すべての関数を発行する前に呼び出す必要があります。

【戻り値】

エラー・コード	値	意味
L_SUCCESS	0x00	初期化成功

※LIN 2.1 ソフトウェア・ドライバでは、L_SUCCESS のみ戻り値として返します。
呼び出し元では、安全のため戻り値のチェックを行うことをお勧めします。

【使用例】

```
/* スタートアップ終了後 */
 if ( l_sys_init() ){
        /* init error */
        ;
 }
 else{
      /* 以後他の LIN2.1 ソフトウェア･ドライバ関数をコール可能 */
 }
```

9.3.2 [マスター] スカラ・シグナル読み出し（3 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 9-4 スカラ・シグナル読み出し関数一覧（マスター）

関数名	機能概要
l_bool_rd	1ビット・スカラ・シグナル読み出し処理
l_u8_rd	8ビット・スカラ・シグナル読み出し処理
l_u16_rd	16ビット・スカラ・シグナル読み出し処理

l_bool_rd

【概要】

1 ビットのスカラ・シグナルを読み出します。

【C 言語記述形式】

l_bool l_bool_rd(l_signal_handle sss)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle sss	シグナル名（マクロ値）

【説明】

この関数は、指定されたシグナル名に関連付けられた、メッセージ・バッファ内のシグナル・データを読み出します。LIN 通信とは非同期に読み出すことが可能です。

【戻り値】

エラー・コード	値	意味
-	0x00,0x01	シグナル・データ

【注意】

・1 ビット以外のサイズで定義したシグナルを読み出さないでください。

・sss は定義したシグナル名以外を指定しないでください。

・送信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_SWITCH
   ・サイズ 1 ビット
*/

if(l_bool_rd( WINDOW_SWITCH ))
{
    “ウィンドウスイッチ ON の場合の処理”
}
else
{
    “ウィンドウスイッチ OFF の場合の処理”
}
```

l_u8_rd

【概要】

1-8 ビットのスカラ・シグナルを読み出します。

【C 言語記述形式】

l_u8 l_u8_rd(l_signal_handle sss)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle sss	シグナル名（マクロ値）

【説明】

この関数は、指定されたシグナル名に関連付けられた、メッセージ・バッファ内のシグナル・データを読み出します。LIN 通信とは非同期に読み出すことが可能です。

新規データの有無に関わらず現在のシグナル・データを読み出します。

【戻り値】

エラー・コード	値	意味
-	0-0xFF	シグナル・データ

【注意】

・8 ビットより大きいサイズで定義したシグナルを読み出さないでください。

・sss には定義したシグナル名以外指定しないでください。

・送信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_STATUS
   ・サイズ 8 ビット
*/

#define ST_BUSY (0x00)
#define ST_IDLE   (0x01)
#define ST_NG                    (0x02)

switch_status = l_u8_rd(WINDOW_STATUS);

if( switch_status == ST_BUSY )
{
        “BUSY 状態の処理”
}
else if( switch_status == ST_NG )
{
        “NG 状態の処理”
}
```

l_u16_rd

【概要】

1-16 ビットのスカラ・シグナルを読み出します。

【C 言語記述形式】

l_u16 l_u16_rd(l_signal_handle sss)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle sss	シグナル名（マクロ値）

【説明】

この関数は、指定されたシグナル名に関連付けられた、メッセージ・バッファ内のシグナル・データを読み出します。LIN 通信とは非同期に読み出すことが可能です。

新規データの有無に関わらず現在のシグナル・データを読み出します。

【戻り値】

エラー・コード	値	意味
-	0-0xFFFF	シグナル・データ

【注意】

・16 ビットより大きいサイズで定義したシグナルを読み出さないでください。

・sss には定義したシグナル名以外指定しないでください。

・送信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_STATUS
   ・サイズ 16 ビット
*/
#define ST_BUSY     0x0000
#define ST_IDLE     0x0001

l_u16 switch_status;

switch_status = l_u16_rd(WINDOW_STATUS);
switch( switch_status )
{
    case ST_BUSY:
        /* BUSY 状態の処理を記述します。 */
        break;
    case ST_IDLE:
        /* IDLE 状態の処理を記述します。 */
        break;
    default:
        /* 異常時の処理を記述します。 */
        break;
}
```

9.3.3 [マスター] スカラ・シグナル書き込み（3 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 9-5 スカラ・シグナル書き込み関数一覧（マスター）

関数名	機能概要
l_bool_wr	1ビット・スカラ・シグナル書き込み処理
l_u8_wr	8ビット・スカラ・シグナル書き込み処理
l_u16_wr	16ビット・スカラ・シグナル書き込み処理

l_bool_wr

【概要】

1 ビットのスカラ・シグナルを書き込みます。

【C 言語記述形式】

void l_bool_wr(l_signal_handle sss, l_bool v)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle　sss	シグナル名（マクロ値）
I	l_bool　　v	書き込みデータ

【説明】

この関数は、指定されたシグナル名に関連付けられたメッセージ・バッファ内に、v で指定したデータを書き込みます。

書き込み後、該当するシグナルの更新フラグをセットします。

【戻り値】

なし

【注意】

・1 ビットより大きいサイズで定義したシグナルを書き込まないでください。

・sss には定義したシグナル以外指定しないでください。

・受信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW、WINDOW_SWITCH
   ・サイズ 1 ビット
*/
#define PUSH      (1)
#define OPEN      (1)

  /* スイッチプッシュを検出 */
if( l_bool_rd(WINDOW_SWITCH) == PUSH )
{
        /* 1 ビット・シグナル OPEN を書き込み */
      l_bool_wr( WINDOW, OPEN );
}
```

l_u8_wr

【概要】

1-8 ビットのスカラ・シグナルを書き込みます。

【C 言語記述形式】

void l_u8_wr(l_signal_handle sss, l_u8 v)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle　sss	シグナル名（マクロ値）
I	l_u8　　v	書き込みデータ

【説明】

この関数は、指定されたシグナル名に関連付けられたメッセージ・バッファ内に、v で指定したデータを書き込みます。

書き込み後、該当するシグナルの更新フラグをセットします。

【戻り値】

なし

【注意】

・8 ビットより大きいサイズで定義したシグナルを書き込まないでください。

・sss には定義したシグナル名以外指定しないでください。

・受信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW、WINDOW_SWITCH
   ・サイズ：8 ビット
*/
#define OPEN      (0x00)
#define CLOSE     (0x01)
#define SLEEP     (0x02)
#define UP        (0x00)
#define DOWN      (0x01)

switch(l_u8_rd(WINDOW_SWITCH))
{
    case UP:
        l_u8_wr( WINDOW, OPEN );
        break;

    case DOWN:
        l_u8_wr( WINDOW, CLOSE );
        break;

    default:
        l_u8_wr( WINDOW, SLEEP);
        break;
}
```

l_u16_wr

【概要】

1-16 ビットのスカラ・シグナルを書き込みます。

【C 言語記述形式】

void l_u16_wr(l_signal_handle sss, l_u16 v)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle sss	シグナル名（マクロ値）
I	l_u16　　v	書き込みデータ

【説明】

この関数は、指定されたシグナル名に関連付けられたメッセージ・バッファ内に、v で指定されたデータを書き込みます。

書き込み後、該当するシグナルの更新フラグをセットします。

【戻り値】

なし

【注意】

・16 ビットより大きいサイズで定義したシグナルを書き込まないでください。この場合、シグナルの 16 ビット分のみ書き込まれます。

・sss に定義したシグナル以外指定しないでください。

・受信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：VOLUME
   ・サイズ 12 ビット
*/

l_u16 vol;

/* ボリュームレベルを取得 */
vol =GetVolLevel( );

/* シグナルに設定 */
l_u16_wr( VOLUME, vol );
```

9.3.4 [マスター] バイト・アレイ読み出し（1 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 9-6 バイト・アレイ読み出し一覧（マスター）

関数名	機能概要
l_bytes_rd	バイト・アレイ読み出し処理

l_bytes_rd

【概要】

バイト・アレイ・データを読み出します。

【C 言語記述形式】

void l_bytes_rd(l_signal_handle sss, l_u8 start, l_u8 count, l_u8* const data)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle sss	シグナル名（マクロ値）
I	l_u8 start	読出し開始バイト番号
I	l_u8 count	読出しバイト数
O	l_u8* const data	読出し格納先

【説明】

この関数は、指定されたシグナル名に関連付けられたメッセージ・バッファ内の、start バイト目から count 数分のバイト値を読み出します。

例えば、バイト・アレイが 0 から 6 で番号付けされた 7 バイト長のメッセージだと仮定します。このバイト・アレイの 3～4 番目から(0、1、2 番目を飛ばして)読み出すには、start に 3 を指定し、count に 2 を指定します。この場合、3 番目の値が data[0]に書き込まれ、4 番目の値が data[1]へ書き込まれます。

新規データの有無にかかわらず、現在のメッセージ・バッファ内のデータを読み出します。

【戻り値】

なし

【注意】

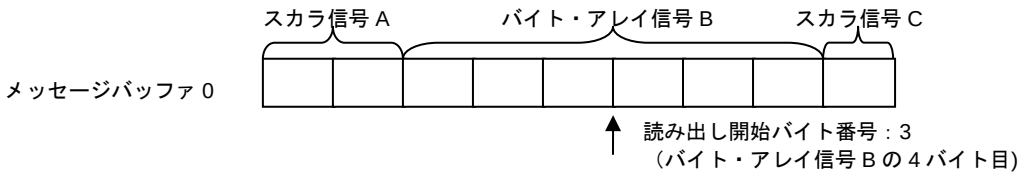
・start と count の組み合わせで、シグナル・サイズ範囲外となる指定をしないでください。正しい値を読み出すことができません。

・バイト・アレイとして定義したシグナルのみ、読み出し可能です（オフセット及びサイズが 8 の倍数となるシグナル）。バイト・アレイでないシグナルを指定した場合、正しい値を読み出すことができません。

・送信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

・読み出し開始バイト番号は、メッセージ・バッファの先頭からの番号ではなく、シグナル名の先頭からの番号になります。下記の例を参照してください。

(例)バイト・アレイ信号 B の 4 バイト目から読み出したい場合



【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_TEST
   ・サイズ：4 バイト
*/

l_u8 data[4];

/* data 配列に 4 バイト分のデータが格納される */
l_bytes_rd(WINDOW_TEST, 0, 4, data)

/* data 配列に 2-3 バイト目までの 2 バイト分のデータが格納される */
l_bytes_rd(WINDOW_TEST, 2, 2, data);
```

9.3.5 [マスター] バイト・アレイ書き込み（1 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 9-7 バイト・アレイ書き込み関数一覧（マスター）

関数名	機能概要
l_bytes_wr	バイト・アレイ書き込み処理

l_bytes_wr

【概要】

バイト・アレイ・データを書き込みます。

【C 言語記述形式】

void l_bytes_wr(l_signal_handle sss, l_u8 start, l_u8 count, const l_u8* const data)

【パラメータ】

I/O	パラメータ	説　　明
I	l_signal_handle　sss	シグナル名（マクロ値）
I	l_u8 start	書き込み開始バイト番号
I	l_u8 count	書き込みバイト数
I	const l_u8* const data	書き込みデータ格納先

【説明】

この LIN2.1 ソフトウェア・ドライバ関数は、指定されたシグナル名に関連付けられた、メッセージ・バッファ内の start バイト目から、count 数分のデータを書き込みます。

例えば、バイト・アレイが 0 から 6 で番号付けされた 7 バイト長のメッセージだと仮定します。このバイト・アレイのバイト 3 と 4 に書き込むには(バイト 0、1、2 を飛ばして)start を 3 にし、(バイト 3 と 4 を書き込む)count を 2 にする必要があります。この場合、バイト 3 は data[0]から読み出した値が書き込まれ、バイト 4 は data[1]から読み出した値が書き込まれます。

書き込み後、該当するシグナルの更新フラグをセットします。

【戻り値】

なし

【注意】

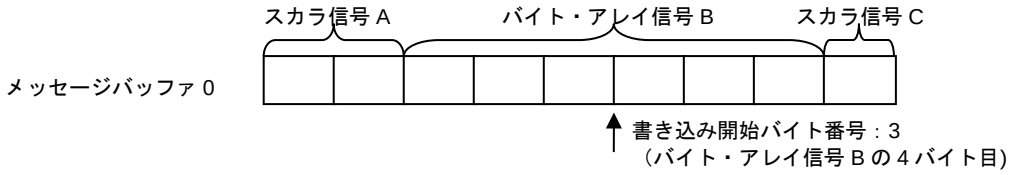
・start と count の組み合わせで、シグナル・サイズ範囲外となる指定をしないでください。正しい値を書き込むことができません。

・バイト・アレイとして定義したシグナルのみ書き込み可能です。（offset 及びサイズが 8 の倍数）バイト・アレイではないシグナルを指定した場合、正しい値を書き込むことができません。

・受信として設定したメッセージ・バッファ内のシグナルを指定しないでください。

・書き込み開始バイト番号は、メッセージ・バッファの先頭からの番号ではなく、シグナル名の先頭からの番号になります。下記の例を参照してください。

(例)バイト・アレイ信号 B の 4 バイト目から書き込みたい場合



【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_TEST
   ・サイズ： 4 バイト
*/
const l_u8 data[4] = {0x00, 0x01, 0x02, 0x03};

/* data 配列の 4 バイト分のデータを書き込む */
l_bytes_wr(WINDOW_TEST, 0, 4, data);

/* WINDOW_TEST シグナルに 2 バイト分のデータを書き込む */
l_bytes_wr(WINDOW_TEST, 2, 2, data);
```

9.3.6 [マスター] 通知（2 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 9-8 通知関数一覧（マスター）

関数名	機能概要
l_flg_tst	シグナル更新フラグ確認
l_flg_clr	シグナル更新フラグクリア

l_flg_tst

【概要】

シグナル更新フラグの値を読み出します。

【C 言語記述形式】

l_bool l_flg_tst(l_flag_handle fff)

【パラメータ】

I/O	パラメータ	説　　明
I	l_flag_handle　fff	フラグ名（シグナル名と同等）

【説明】

この関数は、指定されたフラグ名に関連付けられたシグナル更新フラグを読み出します。

フラグが更新ありに設定されるタイミングは以下の通りです。

・LIN バスからシグナル・データを正常受信し、l_sch_tick 関数をコールした場合

・スカラ・シグナル、バイト・アレイ書き込み関数をコールし、シグナルにデータを書き込んだ場合

【戻り値】

エラー・コード	値	意味
-	0x00	更新なし
-	0x01	更新あり

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_STATUS
   ・サイズ： 1 バイト
*/

l_bool status;

if( l_flg_tst( WINDOW_STATUS ) )
{
        /* WINDOW_STATUS が更新された場合の処理を記載します */
        status = l_bool_rd( WINDOW_STATUS );
}
```

l_flg_clr

【概要】

シグナル更新フラグを 0 クリアします。

【C 言語記述形式】

void l_flg_clr(l_flag_handle fff)

【パラメータ】

I/O	パラメータ	説　　明
I	l_flag_handle　fff	フラグ名（マクロ値、シグナル名と同等）

【説明】

この LIN2.1 ソフトウェア・ドライバ関数は、指定されたフラグ名に関連付けられたシグナルの更新フラグをクリアします。

フラグがクリアされるタイミングは以下の通りです。

・本関数を発行した場合

・LIN バスへシグナル・データを正常送信し、l_sch_tick 関数をコールした場合

【戻り値】

なし

【使用例】

```
/* 事前にコンフィグ・ファイルにて以下の設定にします。
   ・シグナル名：WINDOW_TEST
   ・サイズ：1 バイト
*/

l_u8 status;

if( l_flg_tst(WINDOW_TEST) )
{
      status = l_u8_rd(WINDOW_TEST);

         /* 更新フラグクリア */
      l_flg_clr(WINDOW_TEST);
}
```

9.3.7 [マスター] スケジュール・マネージメント（2 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 9-9 スケジュール・マネージメント関数一覧（マスター）

関数名	機能概要
l_sch_tick	スケジュール制御
l_sch_set	スケジュール設定

l_sch_tick

【概要】

一定間隔ごとに本関数を呼び出すことにより、スケジュール制御を行います。

【C 言語記述形式】

l_u8 l_sch_tick(l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名

【説明】

指定した LIN インタフェースの、現在選択されているスケジュール・テーブルの次のエントリの転送が開始可能かどうかをチェックし、可能であればそのテーブル・エントリの転送を開始します。

次のこの関数のコールで、スケジュール・テーブルの次のエントリの転送が開始可能な場合、その番号を返します。次のエントリの転送が不可能な場合は 0 を返します。

【戻り値】

エラー・コード	値	意味
-	0x00	次のコールでスケジュール・テーブルの次のエントリの転送を開始しません。
-	0x01～0xFF （最大エントリ数以内）	次のコールでスケジュール・テーブルの次のエントリの転送を開始します。値はそのエントリ番号です。

【注意】

・go-to-sleep-command 送信時のシグナル更新は、この関数と同期していません。
go-to-sleep-command の送信が完了したときに、シグナル更新を行います。（go-to-sleep-command の送信が正常終了すると、この関数をコールしなくてもスリープ・モードに移行します）
通常の送受信時は、送受信後、最初の l_sch_tick コールに同期して、シグナル更新が行われます。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください。

・LIN2.1 ソフトウェア・ドライバがスリープ・モードのとき、この関数をコールしないでください。コールした場合、詳細ステータス・ビットの Error in noneactive に 1 がセットされます。

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
*/

/*  タイマ割り込み関数（time_base = 20ms）*/
void main_application_20ms( void )
{

        /* 20ms ごとにスケジュール更新 */
      l_sch_tick ( LIN_CHANNEL0 );
}
```

l_sch_set

【概要】

スケジュール・テーブルをセットします。

【C 言語記述形式】

void l_sch_set(l_ifc_handle iii, L_schedule_handle schedule, l_u8 entry)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名（チャネル番号）
I	l_schedule_handle schedule	スケジュール・テーブル名
I	l_u8 entry	スケジュール・エントリ番号

【説明】

指定した LIN インタフェースのスケジュール・テーブルを schedule で指定したスケジュール・テーブルに切り替えます。

この関数コール後、今のスケジュール・テーブルの次のエントリを転送するタイミング（一回または複数回 l_sch_tick をコールしたとき）で、schedule で指定したスケジュール・テーブルの entry で指定したエントリの転送を開始します。

entry に 0 または 1 をセットすると、schedule の最初のエントリの転送を開始します。

【戻り値】

なし

【注意】

・スケジュール・テーブルのエントリ番号は 1～n（n：スケジュール・テーブルのエントリ数）となります。
　スケジュール・テーブル構造体配列の要素番号（0～n-1）と異なるので注意してください。

・schedule や entry に存在しないスケジュール・テーブル名やエントリ番号を指定しないでください。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください。

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
   ・スケジュール・テーブル名：SCH_TABLE0
*/

/*  タイマ割り込み関数（time_base = 20ms）*/
void main_application_20ms( void )
{
    /* 現在のスケジュール・テーブル・エントリの 3 番が開始されるときに
         スケジュール・テーブル SCH_TABLE0 のエントリ 1 に切り替え
     */
  if( l_sch_tick ( LIN_CHANNEL0 ) == 3 )
  {
    l_sch_set( LIN_CHANNEL0, SCH_TABLE0, 1 );
  }
}
```

9.3.8 [マスター] インタフェース・マネージメント（4 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 9-10 インタフェース・マネージメント関数一覧（マスター）

関数名	機能概要
l_ifc_init	インタフェース初期化
l_ifc_goto_sleep	go-to-sleep-command発行
l_ifc_wake_up	ウェイクアップ発行
l_ifc_read_status	ステータス読み出し

l_ifc_init

【概要】

LIN インタフェースを初期化します。

【C 言語記述形式】

l_bool l_ifc_init (l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名（チャネル番号）

【説明】

この関数は、指定された LIN インタフェースに関連する初期化処理を実施し LIN 通信を使用可能にします。

LIN 送受信の許可処理を実施します。

【戻り値】

エラー・コード	値	意味
L_FAIL	0xFF	失敗
L_SUCCESS	0x00	成功

【注意】

・l_sys_init 関数発行後に使用してください。

・l_sys_init 関数以外の他の LIN2.1 ソフトウェア・ドライバ関数を使用する前に、必ず発行する必要があります。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
*/

if( l_sys_init( ) )
{
    /* エラー処理を記述します */
}
else
{
      /* インタフェース初期化 */
    if( l_ifc_init( LIN_CHANNEL0 ) )
    {
        /* エラー処理を記述します */
    }
}
```

l_ifc_goto_sleep

【概要】

go-to-sleep-command を発行します。

【C 言語記述形式】

void l_ifc_goto_sleep(l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名（チャネル番号）

【説明】

この関数は、指定した LIN バスに go-to-sleep-command を送信します。go-to-sleep-command が正常に送信されると、ステータス・ビットの Goto sleep（l_ifc_read_status 参照）をセットし、スリープ・モードに移行します。go-to-sleep-command の送信に失敗した場合、Goto sleep のセットとスリープ・モードへの移行は行いません。

この関数コールによって go-to-sleep-command を送信するためには、セットされているスケジュール・テーブルに少なくてもひとつのマスター・リクエスト・フレームのエントリが存在しなければなりません。

【戻り値】

なし

【注意】

・go-to-sleep-command は診断コマンドのひとつとして発行します。そのためこの関数をコールする前に ld_is_ready 関数をコールし、LD_SERVICE_IDLE が返されることを必ず確認してください。LD_SERVICE_BUSY が返されるときにこの関数をコールした場合、関数コールが無視されます。

・セットされているスケジュール・テーブルにマスター・リクエスト・フレームのエントリが存在しない場合、go-to-sleep-command は発行されません。その場合はこの関数コール後、l_sch_set 関数をコールし、マスター･リクエスト･フレームのエントリが存在するスケジュール･テーブルに切り替えてください。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください。

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
*/

/* 診断コマンドが受け付け可能であることを確認後、go-to-sleep-command 送信 */
if( ld_is_ready ( LIN_CHANNEL0 ) == LD_SERVICE_IDLE )
{
    l_ifc_goto_sleep( LIN_CHANNEL0 );
}
```

l_ifc_wake_up

【概要】

ウェイクアップを発行します。

【C 言語記述形式】

void l_ifc_wake_up(l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名（チャネル番号）

【説明】

この関数は、指定された LIN インタフェースをスリープ・モードからウェイクアップさせ、LIN バスに 250μs～5ms のウェイクアップ Low パルスを送信します。

【戻り値】

なし

【注意】

・スリープ・モードの場合のみ、発行することが可能です。

・この関数コール後、フレーム転送開始まで一定のウェイト処理が必要です。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
   ・ステータス･マスク･ビット：LD_MASK_SLEEP
*/
  /* スリープ・モードを検出したら、即ウェイクアップ */
if( (l_ifc_read_status(LIN_CHANNEL0) & LD_MASK_SLEEP) == LD_MASK_SLEEP)
{
    l_ifc_wake_up();
}
```

l_ifc_read_status

【概要】

LIN2.1 ソフトウェア・ドライバの各種ステータスを読み出します。

【C 言語記述形式】

l_u16 l_ifc_read_status(l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名（チャネル番号）

【説明】

この関数は、指定された LIN インタフェースのステータス情報を読み出します。

【戻り値】

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit0
最終フレーム PID								0	Save configuration	Event triggered frame collision	Bus activity	Goto sleep	Overrun	Successful transfer	Error in response

最終フレーム ID　：最終通信フレームの PID を示します。

Save configuration　：対応していません。

Event triggered frame collision　：イベント・トリガ・フレームの衝突解決実行中にセットされます。

Bus activity　：バスの立ち上がりエッジまたは立ち下りエッジを検出したときにセットされます。

Goto sleep　：go-to-sleep-command を正常送信した場合にセットされます。

Overrun　：l_ifc_read_status をコールする前に、２回以上の通信が発生した場合：にセットされます。

Successful transfer　：正常に通信が行われた場合セットされます。

Error in response　：レスポンス中に異常通信を検出した場合セットされます。

【注意】

・l_ifc_init 関数発行後に使用してください。

・本関数発行後または l_ifc_init 関数発行後、ステータスは 0 クリアされます。

・ステータスは蓄積フラグとなります。ひとつ前のこの関数コールから通信が複数回行われると、ステータスは上書きされます。

・転送がない状態で連続してこの関数をコールすると、2 回目のコールは 0 を返します。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください。

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
   ・ステータス･マスク･ビット：LD_MASK_ERROR_IN_RESPONSE、LD_MASK_SUCCESSFUL_TRANSFER
*/

status = l_ifc_read_status( LIN_CHANNEL0 );
mask = LD_MASK_ERROR_IN_RESPONSE | LD_MASK_SUCCESSFUL_TRANSFER;

if( status& mask ) == mask )
{
    /* エラーと正常を両方検出、この場合断続的に通信が行われていると考えられる */
}
```

9.3.9 [マスター] ノード・コンフィギュレーション（4 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 9-11 ノード・コンフィグレーション関数一覧（マスター）

関数名	機能概要
ld_is_ready	ノード・コンフィギュレーション・レディ
ld_check_response	ノード・コンフィギュレーション・チェック
ld_assign_frame_id_range	PID割り当て
ld_read_by_id	ID読み出し

ld_is_ready

【概要】

診断コマンドの受信状態をチェックします。

【C 言語記述形式】

l_u8 ld_is_ready (l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle　iii	インタフェース名（チャネル番号）

【説明】

この関数は、指定した LIN インタフェースの診断コマンド（ld_assign_frame_id_range、ld_read_by_id、l_ifc_goto_sleep）が実行可能になったときに LD_SERVICE_IDLE を返します。また、この関数が LD_SERVICE_IDLE を返すとき、前の診断コマンドの結果が有効となります。

この関数が LD_SERVICE_IDLE を返すまでは、他のノード・コンフィギュレーション関数をコールしてはいけません。

【戻り値】

エラー・コード	値	意味
LD_SERVICE_IDLE	0	診断コマンド受付可能
LD_SERVICE_BUSY	1	サービス開始からマスターリクエスト送信完了まで
LD_REQUEST_FINISHED	2	マスターリクエスト送信完了からスレーブ・レスポンス受信完了まで
LD_SERVICE_ERROR	3	バス上でエラーが発生

【注意】

・l_sys_init 関数発行後に使用してください。

・go-to-sleep-command をマスター･リクエスト･フレームとして送信するために、l_ifc_goto_sleep 関数も診断コマンドに含まれます。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください。

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
*/

/*  診断コマンドが受け付け可能であることを確認後、ld_assign_frame_id_range 送信  */
if( ld_is_ready ( LIN_CHANNEL0 ) == LD_SERVICE_IDLE )
{
        ld_assign_frame_id_range ( LIN_CHANNEL0, u1tNad, u1tStartIndex, (u1*)&pids[0]);
}
```

ld_check_response

【概要】

診断コマンドの結果をチェックします。

【C 言語記述形式】

void ld_check_response(l_ifc_handle iii, l_u8* RSID, l_u8* error_code)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle　iii	インタフェース名（チャネル番号）
O	l_u8*　　RSID	最後に受信したRSID
O	l_u8*　　error_code	エラーの詳細情報

【説明】

この関数は、指定した LIN インタフェースの最後に発行したスレーブ･レスポンス･フレームをサポートする診断コマンドの結果を RSID と error_code で返します。error_code の一覧を表 8-12 に示します。erroe_code は関数の戻り値が LD_ERR_NEGRESPONSE のときに有効です。

表 9-12　error_code 一覧

マクロ名	値	意味
LD_ERR_NOERROR	0x00	エラー無し
LD_ERR_NEGRESPONSE	0x01	スレーブ･レスポンス･フレームがネガティブ・レスポンス
LD_ERR_SRFERROR	0x02	スレーブ･レスポンス･フレームのPDUが不正

診断コマンドを転送した結果、スレーブ･ノードが応答しなかったとき（レスポンスを返さなかったとき）にこの関数をコールしても、RSID と error_code に値がセットされます。この場合のセットされる値は、最後にセットされた RSID と error_code となります。また、このときの戻り値も最後に発行したものと同じ結果が返されます。

【注意】

・l_sys_init 関数発行に使用してください。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください。

・本ドライバでは、スレーブ・レスポンス･フレームをサポートする診断コマンドは ld_read_by_id、ld_assign_frame_id_range となります。

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
*/

l_u8 RSID;
l_u8 erroe_code;

/*  診断コマンドが受け付け可能であることを確認後、診断コマンドの結果を取得  */
if( ld_is_ready ( LIN_CHANNEL0 ) == LD_SERVICE_IDLE )
{
        ld_check_response ( LIN_CHANNEL0, &RSID, &error_code );
}
```

ld_assign_frame_id_range

【概要】

スレーブ･ノードのメッセージ･バッファに PID を割り当てます。

【C 言語記述形式】

void ld_assign_frame_id_range (l_ifc_handle iii,l_u8 NAD,l_u8 start_index,const l_u8* const PIDs)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle　iii	インタフェース名（チャネル番号）
I	l_u8　　NAD	NAD値
I	l_u8　　start_index	フレームリストの書き換え開始インデックス値
I	l_8*　　PIDs	PIDリストへのポインタ

【説明】

NAD が一致するノードの最大４つのメッセージフレームの PID を変更または無効にします。

ただし、ID が 60~63(0x3C～0x3F)のフレームは変更できません。

開始インデックスは PID を割り当てる最初のフレームを指定します。フレームの順序はスレーブ・ノードに依存します。リストの最初のインデックスは 0 から開始します。

PID リストの設定値は以下のとおりです。

表 9-13　PID リスト設定値

値	動作
有効なID値(0～59)	指定のPIDに変更
0x00	割り当て解除
0xFF	変更せず以前の値を維持

【戻り値】

なし

【注意】

・この関数をコールする前に ld_is_ready 関数をコールし、LD_SERVICE_IDLE が返されることを必ず確認してください。LD_SERVICE_IDLE 以外が返されるときに、この関数をコールした場合、関数コールが無視されます。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください。

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
*/

/*  診断コマンドが受け付け可能であることを確認後、ld_assign_frame_id_range 送信  */
if( ld_is_ready ( LIN_CHANNEL0 ) == LD_SERVICE_IDLE )
{
        ld_assign_frame_id_range ( LIN_CHANNEL0, u1tNad, u1tStartIndex, (u1*)&pids[0]);
}
```

ld_read_by_id

【概要】

スレーブ･ノードから各種 ID を読み出します。

【C 言語記述形式】

```
void ld_read_by_id(l_ifc_handle iii, l_u8 NAD, l_u16 supplier_id, l_u16 function_id,
                   l_u8 id, l_u8* const data)
```

【パラメータ】

I/O	パラメータ		説　　明
I	l_ifc_handle	iii	インタフェース名（チャネル番号）
I	l_u8	NAD	NAD値
I	l_u16	supplier_id	Supplier ID値
I	l_u16	function_id	Function ID値
I	l_u8	id	ID値
O	l_u8* const	data	データの書き込み先ポインタ

【説明】

NAD、supplier_id、function_id が一致するスレーブ･ノードの id で指定されたプロパティを data で指定された RAM 領域にセットします。スレーブ･ノードが指定された id をサポートしていないときは、否定のレスポンスをセットします。

id と読み出す識別子の関係と否定レスポンス･パターンは　第 5 章　LIN2.1 ソフトウェア・ドライバ概要を参照してください。

【戻り値】

なし

【注意】

・この関数をコールする前に ld_is_ready 関数をコールし、LD_SERVICE_IDLE が返されることを必ず確認してください。

LD_SERVICE_IDLE 以外が返されるときに、この関数をコールした場合、関数コールが無視されます。

・マルチ・チャネル機能未使用時は、インタフェース名に“LIN_CHANNEL0”以外指定しないでください。

【使用例】

```
/* コンフィグ・ファイルにて以下定義済み
   ・チャネル名：LIN_CHANNEL0
*/

/*  診断コマンドが受け付け可能であることを確認後、ld_read_by_id 送信  */
if( ld_is_ready ( LIN_CHANNEL0 ) == LD_SERVICE_IDLE )
{
    ld_read_by_id( LIN_CHANNEL,
                   u1tNad,                  /* NAD     */
                   u2tSupplierId,           /* SupID   */
                   u2tFunctionId,           /* FuncID */
                   u1tId,                   /* ID      */
                   (u1 *)&ApLst_u1sData ); /* Data    */
}
```

9.3.10 [マスター] ユーザ定義コール・アウト（4 種類）

ここでは、以下の表に示すLIN2.1ソフトウェア・ドライバ関数について説明します。

表 9-14 ユーザ定義コール・アウト関数一覧（マスター）

関数名	機能概要
l_sys_irq_disable	割り込み禁止設定
l_sys_irq_restore	割り込み復帰設定
l_sys_call_wake_up	ウェイクアップ開始
l_sys_call_fatal_error	致命的エラー解決

l_sys_irq_disable

【概要】

割込みを禁止します。

【C 言語記述形式】

l_irqmask l_sys_irq_disable(void)

【パラメータ】

なし

【説明】

この関数は割込みを禁止する為のコール・アウト関数です。

DI を実行し、現在の割込み状態を返却します。返却値は、CC-RL 版のとき PSW の値、IAR 版のとき IAR 組み込み関数__get_interrupt_state によって返却された値です。

【戻り値】

I/O	値	意味
—	[CC-RL] PSWの割り込み状態ビット [IAR] グローバル割り込み状態	

【注意】

本関数はコール・アウト関数なので変更可能ですが、スマート・コンフィグレータで提供されるコードから変更しないでください。（「5. 2. 7 ユーザ定義コール・アウトの実装」参照）

【使用例】

自動的にコールされる為、アプリケーション内で使用する必要はありません。

l_sys_irq_restore

【概要】

割込み状態を元に戻します。

【C 言語記述形式】

void l_sys_irq_restore(l_irqmask previous)

【パラメータ】

I/O	パラメータ	説　　明
I	l_irqmask previous	割り込み復帰時の状態値 （[CC-RL] PSWの割り込み状態ビット [IAR] グローバル割り込み状態）

【説明】

この関数は、割込み状態を復帰させる為のコール・アウト関数です。

デフォルトでは、CC-RL 版のとき previous が割り込み許可設定（0x80）の場合 EI に設定し、それ以外は DI に設定します。IAR 版のときは IAR 組み込み関数__set_interrupt_state により、previous の指定状態に復帰します。

【戻り値】

なし

【注意】

本関数はコール・アウト関数なので変更可能ですが、スマート・コンフィグレータで提供されるコードから変更しないでください。（「5. 2. 7 ユーザ定義コール・アウトの実装」参照）

【使用例】

自動的にコールされる為、アプリケーション内で使用する必要はありません。

l_sys_call_wake_up

【概要】

Wakeup 受信直後にコールされます。

本ソフトウェア独自の拡張関数です。

【C 言語記述形式】

void l_sys_call_wake_up(l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名

【説明】

この関数は、LIN2.1 ソフトウェア・ドライバがウェイクアップを受信した場合に、コール・アウトされる関数です。

【戻り値】

なし

【注意】

本関数の実装は必須ではありません。使用しない場合は空処理としてください。スマート・コンフィグレータで提供されるコードでは、空処理の関数となっています。（「5. 2. 7 ユーザ定義コール・アウトの実装」参照）

【使用例】

自動的にコールされる為、アプリケーション内で使用する必要はありません。

l_sys_call_fatal_error

【概要】

自ハードウェアに起因するエラーを検出した場合、ドライバによってコールされます。

本ソフトウェア独自の拡張関数です。

【C 言語記述形式】

void l_sys_call_fatal_error(l_ifc_handle iii)

【パラメータ】

I/O	パラメータ	説　　明
I	l_ifc_handle iii	インタフェース名

【説明】

この関数は、ドライバ内で解決不可能なエラーを検出した場合に、ユーザに通知を行う関数です。

関数コール時にはドライバはバスから切断され、RAM が初期化された状態になっています。

l_ifc_init 関数をコールしてバスに再接続してください。l_ifc_init 関数が失敗した場合、必要に応じてリトライしてください。

【戻り値】

なし

【注意】

スマート・コンフィグレータで提供される本関数のコードでは、l_ifc_init 関数が失敗した場合の処理がないため、必要に応じてリトライを追加してください。（「5. 2. 7 ユーザ定義コール・アウトの実装」参照）

【使用例】

自動的にコールされる為、アプリケーション内で使用する必要はありません。

第10章 LIN 記述ファイル（LDF）の記述例

LIN記述ファイル（LDF）は、LINネットワークを定義する標準形式です。
LDFでは、グローバル定義、ノード定義、シグナル定義、フレーム定義、スケジュール定義などのLIN通信に必要となる情報を定義します。

以下にLDFの記述例を示します。
本例では、分類ごとに記述例を示していますが、実際には1つのファイル内ですべて定義されます。

分類	記述例
グローバル情報定義 （LINプロトコルバージョン、通信速度など）	LIN_description_file; LIN_protocol_version = "2.1"; LIN_language_version = "2.1"; LIN_speed = 9.6 kbps;

ノード定義
（ノード名、タイム・ベースなど）
	Nodes {
	  Master: LIN_Master, 10 ms, 0.2 ms ;
	  Slaves: LIN_Slave ;
	}

シグナル定義
（シグナル名、サイズ、初期値、送信/受信ノードなど）
	Signals {
	  Sample_PublishSig1: 10, 0, LIN_Master, LIN_Slave ;
	  Sample_PublishSig2: 3, 0, LIN_Master, LIN_Slave ;
	  Sample_PublishSig3: 3, 0, LIN_Master, LIN_Slave ;
	  Sample_SubscribeSig1: 10, 0, LIN_Slave, LIN_Master ;
	  Sample_SubscribeSig2: 3, 0, LIN_Slave, LIN_Master ;
	  Sample_SubscribeSig3: 2, 0, LIN_Slave, LIN_Master ;
	  RsErr: 1, 0, LIN_Slave, LIN_Master ;
	}

分類	記述例

診断シグナル定義	Diagnostic_signals {
	MasterReqB0: 8, 0 ;
	MasterReqB1: 8, 0 ;
	MasterReqB2: 8, 0 ;
	MasterReqB3: 8, 0 ;
	MasterReqB4: 8, 0 ;
	MasterReqB5: 8, 0 ;
	MasterReqB6: 8, 0 ;
	MasterReqB7: 8, 0 ;
	SlaveRespB0: 8, 0 ;
	SlaveRespB1: 8, 0 ;
	SlaveRespB2: 8, 0 ;
	SlaveRespB3: 8, 0 ;
	SlaveRespB4: 8, 0 ;
	SlaveRespB5: 8, 0 ;
	SlaveRespB6: 8, 0 ;
	SlaveRespB7: 8, 0 ;
	}

フレーム定義
（フレーム名、ID、サイズ、シグナル名など）
	Frames {
	Sample_Frame1: 10, LIN_Master, 8 {
	Sample_PublishSig1, 0 ;
	Sample_PublishSig2, 16 ;
	Sample_PublishSig3, 24 ;
	}
	Sample_Frame2: 20, LIN_Slave, 8 {
	Sample_SubscribeSig1, 0 ;
	Sample_SubscribeSig2, 10 ;
	Sample_SubscribeSig3, 16 ;
	RsErr, 63 ;
	}
	}

分類	記述例

診断フレーム定義	Diagnostic_frames {
	  MasterReq: 0x3c {
	  MasterReqB0, 0 ;
	  MasterReqB1, 8 ;
	  MasterReqB2, 16 ;
	  MasterReqB3, 24 ;
	  MasterReqB4, 32 ;
	  MasterReqB5, 40 ;
	  MasterReqB6, 48 ;
	  MasterReqB7, 56 ;
	  }
	  SlaveResp: 0x3d {
	  SlaveRespB0, 0 ;
	  SlaveRespB1, 8 ;
	  SlaveRespB2, 16 ;
	  SlaveRespB3, 24 ;
	  SlaveRespB4, 32 ;
	  SlaveRespB5, 40 ;
	  SlaveRespB6, 48 ;
	  SlaveRespB7, 56 ;
	  }
	}

ノード属性定義
（LINプロトコルバージョン、製品識別番号など）
	Node_attributes {
	  LIN_Slave{
	  LIN_protocol = "2.1" ;
	  configured_NAD = 0xA ;
	  initial_NAD = 0xA ;
	  product_id = 0x6E, 0x3039, 0 ;
	  response_error = RsErr ;
	  P2_min = 50 ms ;
	  ST_min = 0 ms ;
	  N_As_timeout = 1000 ms ;
	  N_Cr_timeout = 1000 ms ;
	  configurable_frames {
	  Sample_Frame1 ;
	  Sample_Frame2 ;
	  }
	  }
	}

分類	記述例

スケジュール・テーブル定義
（スケジュール・テーブル名、フレーム名、送信間隔など）
	Schedule_tables {
	Master_Req {
	MasterReq delay 20 ms ;
	}
	Slave_Resp {
	SlaveResp delay 20 ms ;
	}
	Normal_1 {
	Sample_Frame1 delay 20 ms ;
	}
	}

付録　改版履歴

版数	内容	適用箇所
Rev.1.00	LIN2.0, 2.1 ソフトウェア・ドライバ ユーザーズ・マニュアル Rev.4.80を基に作成	全般

ユーザーズ・マニュアル

発行年月日　2022 年 4 月 20 日　Rev 1.00

発行　ルネサス エレクトロニクス株式会社
〒135-0061 東京都江東区豊洲三丁目 2-24

RLIN3 モジュール
Software Integration System

RENESAS
ルネサスエレクトロニクス株式会社